University of Modena and Reggio Emilia

XXXIV cycle of the International Doctorate School in Information and Communication Technologies

Doctor of Philosophy dissertation in Computer Engineering and Science

Perception, Reasoning, Action:

The New Frontier of Embodied AI

Federico Landi

Supervisor: Prof. Rita Cucchiara

PhD Course Coordinator: Prof. Sonia Bergamaschi

Modena, 2022

Contents

Chapter 1

Introduction

After the Deep Learning Revolution occurred in the last decade, our world has become increasingly interactive, digitalized, and intelligent. Nowadays, some complex models can understand what we say and answer accordingly. Other architectures can recognize President Obama or a movie poster from Harry Potter among thousands of different images. However, the actual embodiment of artificial intelligence is still far from being achieved. The field of Robotics is making remarkable progress, with the creation of robots able to run, dance, and perform somersaults and impressive parkour moves. Nevertheless, the presence of intelligent collaborative agents in our homes is an ambitious yet distant goal.

Why is that so? Different from humans, machines can elaborate a vast amount of data. As a result, they can compute nearly perfect solutions on precise and well-defined tasks. But while the data processing abilities of machines are unmatched by humans, artificial intelligence lacks interpretation, imagination, and common sense. For instance, finding a spoon in a kitchen is a non-trivial task for a robot. As a first problem, the agent might not know what spoons or kitchens are. Second, spoons are generally inside drawers. While humans make this semantic connection automatically thanks to their experience and can perform this action with ease, opening a drawer is a complex interaction with the environment that the agent might not even know. Together with these challenges, there are many other obstacles that a robotic assistant would face. To accomplish the embodiment of artificial intelligence inside robots, we must equip the machines with the instruments to perceive, reason, and act in the world around them, our everyday world.

1.1 Motivation

Recent advances in Deep Learning have allowed the creation of complex architectures that can deal with many different tasks. For instance, Computer Vision models can identify the content of an image and detect the individual elements with unprecedented accuracy. At the same time, progress in the Natural Language Processing field yields impressive results. We can all experience it in our everyday life: we can ask the vocal assistant on our mobile phone to find a picture of an Indri on the internet. In this simple way, we can all understand how much the processing abilities of our devices have improved in the last few years. While the results achieved by the vision-and-language communities are impressive, there is still a key component missing in these architectures: autonomy. Indeed, they cannot move or decide what to look at, and as such, they cannot experience the world as we all know it. As we try to enable these behaviors in the next generation of artificial intelligence models, we face the great challenges of embodiment [26, 62, 173, 174] and Embodied Artificial Intelligence (Embodied AI) [54]. We define:

Definition 1.1.1 (Embodied AI)**.** *Research field that strives for the creation of embodied agents which learn, through interaction and exploration, to solve challenging tasks within their environments.*

When we include the processing abilities of modern deep learning architectures for vision and language in a robot that perceives and acts in an environment, we face two different challenges. The first challenge is related to the concept of time [53]. We all know that every action has a consequence and produces results in the future. For instance, opening a drawer will allow us to pick up items from inside it but could hinder the passage later. We can account for the effects of our actions in the long term because we have been dealing with them since the day we were born. However, long-term dependencies create many troubles for modern intelligent systems [20]. We define:

Definition 1.1.2 (Long-term Dependencies)**.** *Cause-effect relationships between two or more actions taken in different (generally distant) moments in time.*

The second challenge arises from the presence of information coming from different domains [120]. Such domains are vision, language, and action (see Fig. 1.1). The vision domain includes what the agent sees with its sensory suite. The language domain comprises textual information, descriptions, and instructions. The language domain is usually where human-robot communication happens in the form of orders and feedbacks. Finally, the action domain is peculiar to robots and contains position, velocity, and other information about the interactions between the agent and the environment. While there are great results on tasks involving single modalities and recently also on vision-and-language applications, there is

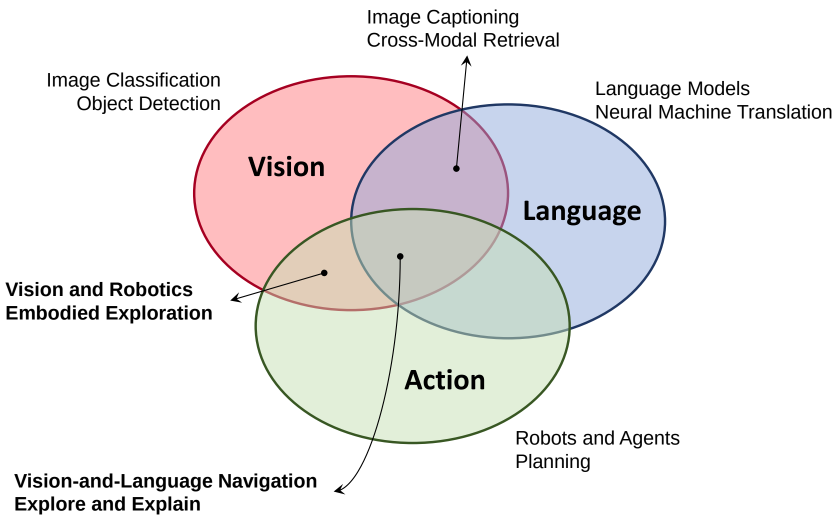


Figure 1.1: Embodied AI comprises three main different modalities. The tasks tackled in this thesis lie at the intersection of vision, action, and language.

still much ongoing research trying to solve challenges related to the fusion of all these three modalities [153, 154]. By using the robot perspective, we define:

Definition 1.1.3 (Multimodality)**.** *The capacity for an agent to perform a task in the environment by combining information coming from at least two different sensory channels.*

In this thesis, we focus on these two main challenges. Dealing with long-term dependencies and multimodality is crucial for embodied agents. As such, we provide numerous insights and experimental evidence as to what techniques are more effective when dealing with different tasks in the new field of Embodied AI.

1.2 Problem Statement

Human beings and machines have very different ways to perform simple tasks in their environment. Similarly, their respective efficacy is tremendously distant, with human performance being much higher than the one achieved by robotic agents. For the most part, this difference in performance is due to the different ways in which humans and machines deal with long-term dependencies and multimodality. It is no doubt that if we are to achieve nearly-human performance in Embodied AI, we must address these two aspects. While reproducing either the ability or the complex structure of the human brain is a dream achievable purely by science fiction, we can try to minimize the performance gap for simple tasks such as exploration and navigation in an environment between a human and their robotic counterpart. Hence, the fundamental question of this thesis is:

How to bridge the performance gap for simple Embodied AI tasks?

Our efforts to minimize the discrepancy between human and artificial performance begin with an analysis of the role of time. Traditional methods for vision-and-language have experienced the difficulties of dealing with time series. For instance, to generate a caption for an image or video, one must be aware of the concept of time. In other words, the reasoning models need to know what is *before* and what is *after*. The same applies to Embodied AI, but on a completely different scale. The agent must not only *reason* about time but also *perceive* and *act* inside time. During the chain of events represented by a navigation episode, the observations made by the agent greatly depend on its previous decisions and actions. The first question arising when dealing with simple Embodied AI tasks is:

How to deal with time series with long-term dependencies?

Immediately after, the newborn agent needs a simple yet effective way to perceive the environment and act inside it. The action of exploring an unknown area is a simple form of interaction between the agent and its environment. For this reason, in this thesis, we consider the task of embodied exploration as a crucial testbed for our agents. The second question in our journey in Embodied AI is:

How to merge visual perception and actions to enable exploration?

While exploration is conceptually simple, it can enable a series of more complex downstream tasks. For instance, given an appropriate architecture for the agent, it is possible to employ the knowledge gathered in exploration to perform goal-driven navigation. In the same way, it is possible to influence the learning process with task-specific rewards to teach the agent different simple tasks. The third question is:

How to combine learning methods and architectural modules to enable new tasks?

Finally, we want our agent to understand our instructions. Therefore, we need to include the language modality in the loop. In our specific case, we want the agent to interpret textual instructions. Examples of such instructions may be "*turn*

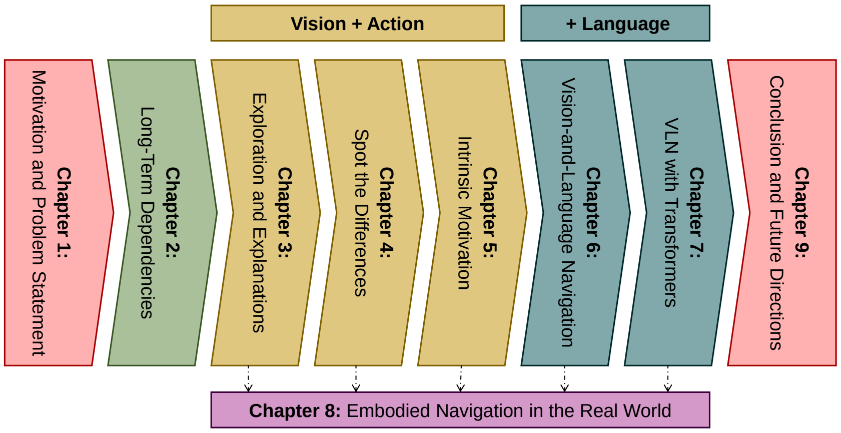

Vision + Action
+ Language
Chapter 1: Motivation and Problem Statement
Chapter 2: Long-Term Dependencies
Chapter 3: Exploration and Explanations
Chapter 4: Spot the Differences
Chapter 5: Intrinsic Motivation
Chapter 6: Vision-and-Language Navigation
Chapter 7: VLN with Transformers
Chapter 9: Conclusion and Future Directions
Chapter 8: Embodied Navigation in the Real World

Figure 1.2: Outline of the work addressed in this thesis.

right, get out of the kitchen, and move towards the red armchair. Stop there and wait". To execute this kind of order, it is evident that the robot needs to ground linguistic descriptions into its visual observations. In our journey in Embodied AI, the fourth and last question is:

How to include natural language in the agent reasoning system to enable language-driven visual navigation?

Answering the four questions will help us reduce the performance gap between human and robotic performance on these simple tasks. And even after, progress will be possible mainly by finding new, better solutions and answers to these four questions.

The field of Embodied AI has seen a proliferation of new tasks and methods in recent years. Similarly, it has witnessed an increase in the availability and diversity of simulating platforms. Unfortunately, this diversity hinders reproducibility and substantial improvements. The introduction of the Habitat simulating platform [145] strives to counter this trend and provide a common research ground for Embodied AI. We joined this initiative by evaluating a substantial part of our work on the Habitat platform.

1.3 Organization

This thesis presents a step-by-step approach to the new and complex challenges of Embodied AI. Fig. 1.2 depicts the overall structure of this process.

In Chapter 2, we discuss Recurrent Neural Networks (RNNs). RNNs are the most common approach when dealing with time series. In particular, Long Short-Term Memory (LSTM) is the standard de-facto for many tasks involving sequential inputs and long-term dependencies. As such, they represent an enabling technology for Embodied AI. We introduce a heuristic enhancement of LSTM that brings better results, increased training stability, and reduced convergence time on a set of tasks.

Chapter 3 introduces the heart of the thesis. We place the embodied agent in a simulated photorealistic unknown environment and teach the robot to explore it. To that end, we propose an approach relying on artificial curiosity, where the agent tries to maximize its surprisal during the exploration episode. While exploring, the agent must produce natural language descriptions of what it sees. We call this setting *Explore and Explain*. In Chapter 4, we devise a new task involving embodied exploration where the agent already knows the environment layout, but something has changed since its last episode. The goal is to recognize and identify as many differences as possible in a given time window. We name this new task *Spot the Difference*. Chapter 5 introduces a modular architecture for embodied exploration trained with a purely intrinsic reward. Our strategy promotes actions likely to produce a high impact on the environment. We then show that exploration is an essential ability of embodied agents and that it can enable a series of downstream tasks such as coordinate-driven navigation in unknown environments.

Then, in Chapters 6 and 7, we tackle the recent task of Vision-and-Language Navigation (VLN). In VLN, the agent needs to follow a language-specified instruction to reach a target location in a new environment. We design two different methods to fuse lingual and visual information. In Chapter 6, we present an approach based on dynamic convolutional filters. This method exploits the instruction and internal state of the policy LSTM to fuse information coming from the visual and textual modality via dynamic convolution. In Chapters 7, we devise a fully-attentive architecture for VLN. Taking inspiration from recent advances in Natural Language Processing and Vision-and-Language, we design a Transformer-like architecture for VLN. Self-attention can deal with long-term dependencies effectively. As such, it is a convenient choice for our case study. At the same time, multimodality is achieved by design thanks to cross-attention layers. We call the resulting architecture *Perceive, Transform, and Act* (PTA). In this last part of the thesis on Vision-and-Language Navigation, we show that it is possible to include natural language instructions from a human user in the agent reasoning motor. Hence, we enable a series of future research directions and applications.

As an additional contribution, we discuss the deployment of agents trained in simulation in the real world. We present this discussion in Chapter 8. While most of our experiments exploit simulation, we show that it is possible to deploy the resulting models on a Low-Cost Robot (LoCoBot) [106] with little effort. The work presented in Chapter 8 is orthogonal to the material outlined in the previous Chapters and represents a further research direction. This direction will need more and more consideration as Embodied AI progresses towards more intelligent and autonomous agents. Chapter 9 depicts these possible future directions and presents the conclusion of this thesis.

Chapter 2

Working Memory Connections for LSTM

Recurrent Neural Networks with Long Short-Term Memory (LSTM) make use of gating mechanisms to mitigate exploding and vanishing gradients when learning long-term dependencies. For this reason, LSTMs and other gated RNNs are widely adopted, being the standard *de facto* for many sequence modeling tasks. Although the memory cell inside the LSTM contains essential information, it is not allowed to influence the gating mechanism directly. In this Chapter, we improve the gate potential by including information coming from the internal cell state. The proposed modification, named Working Memory Connection, consists in adding a learnable nonlinear projection of the cell content into the network gates. This modification can fit into the classical LSTM gates without any assumption on the underlying task, being particularly effective when dealing with longer sequences. Previous research effort in this direction, which goes back to the early 2000s, could not bring a consistent improvement over vanilla LSTM. As part of our work, we identify a key issue tied to previous connections that heavily limits their effectiveness, hence preventing a successful integration of the knowledge coming from the internal cell state. We show through extensive experimental evaluation that Working Memory Connections constantly improve the performance of LSTMs on a variety of tasks. Numerical results suggest that the cell state contains useful information that is worth including in the gate structure.

This Chapter is related to publication [8], as reported in the List of Publications.

2.1 Introduction

Recurrent Neural Networks (RNNs) [53, 142] are a family of architectures that process sequential data by means of internal hidden states. The set of parameters of the network is shared across time steps, allowing the RNN to process inputs of variable length. As RNNs suffer from the so-called exploding and vanishing gradient problem (EVGP) [19, 77], which hinders the learning of long-term dependencies [20, 128], previous works have proposed to enrich the recurrent cell with gating mechanisms [78, 84]. For instance, Long Short-Term Memory networks (LSTMs) [78] use gates to control the information flow towards and from the memory cell and to regulate the forgetting process [61]. LSTMs are adopted in a wide number of tasks, such as neural machine translation [13, 158], speech recognition [68], and also vision-and-language applications like image and video captioning [15, 166, 177].

In this Chapter, we propose a novel cell-to-gate connection that modifies the classic LSTM block. Our formulation is general and improves LSTM overall performance and training stability without any particular assumption on the underlying task. In the *vanilla* LSTM formulation, the gates are controlled by the current input of the block and its previous output, which acts as the hidden state for the network. The long-term memory cell, instead, is employed to store information during the forward pass and provides a safe path for back-propagating the error signal. We argue that the content stored in the memory cell could be useful to regulate the gating mechanisms, too. The key element of our design is a connection between the memory cell and the gates with a protection mechanism that prevents the cell state from being exposed directly. We draw inspiration from the gated *read* operation employed to reveal the cell content at the block output, and enrich it with a learnable projection. In this way, the LSTM block can use the knowledge in the cell (acting as a long-term memory) to control the evolution of the whole network in the short-term.

A similar concept in cognitive psychology and neuroscience is the so-called *working memory* [55], a type of memory employed, for instance, *to retain the partial results while solving an arithmetic problem without paper, or to combine the premises in a lengthy rhetorical argument* [75]. Although definitions are not unanimous, working memory is said to be a cognitive system acting as a third type of memory between long-term and short-term memory. Our connections share this characteristic with working memory. For this reason, we call them *Working Memory Connections* (WMCs).

A first attempt to fuse the information of the cell in the gates was made with the design of *peepholes* [60]: direct multiplicative connections between the memory cell and the gates. This approach has not been largely adopted in literature, as recent studies report mixed results [69] and discourage their use. Since our idea recalls the rationale of peephole connections, we provide a large comparison with this previous work. By doing so, we point out the major issues in the peephole formulation that hinder effective learning and attest that WMCs do not suffer from the same problems. In our experiments, we show that an LSTM equipped with Working Memory Connections achieves better results than comparable architectures, thus reflecting the theoretical advantages of their design. In particular, WMCs surpass vanilla LSTM and peephole LSTM in terms of final performances, stability during training, and convergence time. All these aspects testify the advantage in letting the cell state participate in the gating dynamics. In order to support our conclusions, we conduct a thorough experimental analysis covering a wide area of current research topics.

To sum up, our contribution is mainly three-fold. First, we present a modification of LSTM in which traditional gates are enriched with Working Memory Connections, linking the memory cell with the gates through a protection mechanism. Then, we demonstrate that exposing the LSTM internal state directly and without a proper protection yields unstable training dynamics that compromise the final performance. Finally, we show the effectiveness of the proposed solution in a variety of tasks, ranging from toy problems with very long-term dependencies (adding problem, copy task, and sequential MNIST) to language modeling and image captioning.

2.2 Related Work

Long Short-Term Memory networks [78] aim to mitigate the exploding and vanishing gradient problem [20, 77] with the use of gating mechanisms. Since its introduction, LSTM has gained a lot of attention for its flexibility and efficacy in many different tasks. To simplify the LSTM structure, Liu *et al.* [105] propose to exploit the content of the long-term memory cell in a recurrent block with only two gates. However, this model neglects the importance of the LSTM output. While this might be useful for simple tasks, it is unlikely to generalize to more complex settings. Arpit *et al.* [12] propose to modify the path of the gradients in order to stabilize training with a stochastic algorithm specific to LSTM optimization. This direction of work is not in contrast with our goal, and could possibly be integrated with our proposal since our connection does not require a specific setup to be optimized. Among the LSTM variants, the Gated Recurrent Unit (GRU) [37, 38] is the most popular and common architecture [40], and features a coupling mechanism between input and forget gates [69]. A recent line of research aims to tailor the LSTM structure for specific tasks. For instance, Baraldi *et al.* [15] propose a hierarchical model for video captioning, while other works incorporate convolutional models into the LSTM structure [100, 176]. While these works propose a modification of the LSTM towards a specific goal, we propose a general and powerful idea that adapts to a large set of different tasks.

Recently, models based on self-attention, such as Transformer [164] and its variants, are achieving state-of-art performances on many different tasks, and also for sequence modeling. For instance, language representations based on BERT [51] can be finetuned with an additional output layer to obtain state-of-art results on many language-based tasks. However, RNNs require much fewer parameters and operations to run than Transformer-based architectures and are still widely adopted. Moreover, LSTMs still have a large market in embedded systems and edge devices for their low computational and memory requirements.

2.3 Proposed Method

In this Section, we present a complete overview of Working Memory Connections. First, we recall the LSTM equations. Second, we explain the modifications introduced in our design. Finally, we motivate the choices behind WMCs *w.r.t.* other approaches. Specifically, we identify key problems in previous cell-to-gate connections that hinder the learning process, and we show that the proposed solution does not suffer from these weaknesses.

2.3.1 LSTM

The core idea behind Long Short-Term Memory networks is to create a constant error path between subsequent time steps. Being $\mathbf{x}_t$ the input vector at time t we can write the rollout equations for a vanilla LSTM as:

$$\mathbf{g}_t = \tanh(\mathbf{W}_{gx}\mathbf{x}_t + \mathbf{W}_{gh}\mathbf{h}_{t-1} + \mathbf{b}_g) \tag{2.1}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \tag{2.2}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \tag{2.3}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \tag{2.4}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \tag{2.5}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \tag{2.6}$$

Here, $\mathbf{g}$ is the block input, $\mathbf{i}$, $\mathbf{f}$, and $\mathbf{o}$ are respectively the input, forget, and output gates, $\mathbf{c}$ represents the memory cell value, and $\mathbf{h}$ is the block output. In this notation, σ is the sigmoid function and $\odot$ denotes element-wise Hadamard product. In its first formulation [78], LSTM did not include the multiplicative forget gate. However, being able to forget about past inputs [61] allows LSTM to tackle longer sequences while not hindering the back-propagation of the error signal.

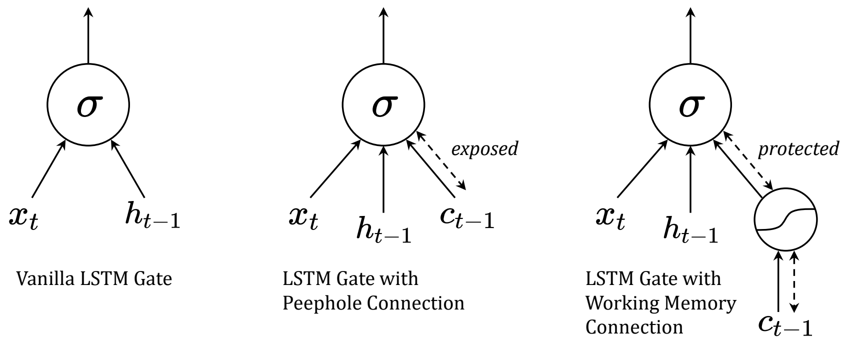


Figure 2.1: Comparison between a vanilla LSTM gate, a peephole connection, and a Working Memory Connection.

2.3.2 Working Memory Connections

In the following, we introduce Working Memory Connections, which enable the memory cell to influence the value of the gates through a set of recurrent weights. Given a proper design for the connection, we argue that there is a practical advantage in letting the cell state influence the gating mechanisms in the LSTM block directly. In fact, the cell state $\mathbf{c}_t$ provides unique information about the previous time steps that are not present in $\mathbf{h}_t$. For instance, $\mathbf{h}_t$ may be close to zero as a consequence of the output gate saturating towards zero (see Eq. 2.6), while $\mathbf{c}_t$ may be growing and changing as a result of a sequence of input vectors. In that case, since the cell state cannot control the output gate, the LSTM block is forced to learn which particular value in the input vector is the marker that signals to open the output gate. Instead, with an appropriate connection strategy, the LSTM block could learn a mapping between the cell internal state and the gate values.

Our solution employs a set of recurrent weights $W_{\star c}$, $\star \in \{\mathbf{i}, \mathbf{f}, \mathbf{o}\}$ and a nonlinear activation function to model a connection between memory cell and gates. The application of a non-linearity on the memory cell is coherent with the present LSTM structure: as it can be noticed from Eq. 2.6, a nonlinear activation function is applied to $\mathbf{c}_t$ before the Hadamard product with $\mathbf{o}_t$[1]. In light of the above-mentioned intuitions, we modify Eq. 2.2, 2.3, and 2.5 by exposing the cell

[1]Previous works [69] have also shown that removing this non-linearity leads to a significant loss in terms of performance.

state $\mathbf{c}_t$ at time t through a protection mechanism as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \tanh(\mathbf{W}_{ic}\mathbf{c}_{t-1}) + \mathbf{b}_i) \tag{2.7}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \tanh(\mathbf{W}_{fc}\mathbf{c}_{t-1}) + \mathbf{b}_f) \tag{2.8}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \tanh(\mathbf{W}_{oc}\mathbf{c}_t) + \mathbf{b}_o), \tag{2.9}$$

where $\mathbf{W}_{\star c}\mathbf{c}_t$ denotes a general linear transformation.

At a first glance, Working Memory Connections may seem redundant in the gate structure. In fact, h_{t-1} depends from the value of c_{t-1} (Eq. 2.6). This impression is misleading, as the proposed connections introduce two main aspects of novelty. First, the non-linear activation function operates on three different projections of the cell state, one for each gate type. Second, Eq. 2.9 shows that the connection on the output gate depends on c_t, rather than on c_{t-1}, hence allowing for a more responsive control of the output dynamics of the entire LSTM block.

2.3.3 Advantages of Working Memory Connections

To formally motivate the improvement given by Working Memory Connections, we start by considering the local gradients of the gates in which the cell interaction is added. We limit our formal analysis to the input gate $\mathbf{i}_t$, but our reasoning can be generalized to $\mathbf{f}_t$ and $\mathbf{o}_t$. If we denote by $\bar{\mathbf{i}}_t$ the argument of the sigmoid activation function (Eq. 2.7) at time t:

$$\bar{\mathbf{i}}_t = \mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \tanh(\mathbf{W}_{ic}\mathbf{c}_{t-1}) + \mathbf{b}_i, \tag{2.10}$$

then the local gradient of the input gate $\mathbf{i}_t$ is expressed by:

$$\frac{\partial \mathbf{i}_t}{\partial \bar{\mathbf{i}}_t} = \frac{\partial}{\partial \bar{\mathbf{i}}_t}\sigma\left(\bar{\mathbf{i}}_t\right) = \textit{diag}\left[\sigma\left(\bar{\mathbf{i}}_t\right) \odot \left(\mathbf{1} - \sigma\left(\bar{\mathbf{i}}_t\right)\right)\right], \tag{2.11}$$

where $\mathbf{1}$ denotes a vector of ones, and *diag*$[\mathbf{x}]$ indicates a diagonal N $\times$ N matrix whose diagonal contains the N elements of vector $\mathbf{x}$.

From here, we can easily derive the local gradients on the recurrent weights $\mathbf{W}_{ix}$, $\mathbf{W}_{ih}$, and $\mathbf{W}_{ic}$ at time t:

$$\frac{\partial \mathbf{i}_t}{\partial \mathbf{W}_{ix}} = \frac{\partial \mathbf{i}_t}{\partial \bar{\mathbf{i}}_t} \otimes \mathbf{x}_t, \tag{2.12}$$
$$\frac{\partial \mathbf{i}_t}{\partial \mathbf{W}_{ih}} = \frac{\partial \mathbf{i}_t}{\partial \bar{\mathbf{i}}_t} \otimes \mathbf{h}_{t-1}, \tag{2.13}$$
$$\frac{\partial \mathbf{i}_t}{\partial \mathbf{W}_{ic}} = \delta\hat{\mathbf{i}}_t \otimes \mathbf{c}_{t-1}, \tag{2.14}$$

where $\otimes$ denotes the outer product of two vectors, and:

$$\delta\hat{\mathbf{i}}_t = \frac{\partial \mathbf{i}_t}{\partial \bar{\mathbf{i}}_t} \odot \left(\mathbf{1} - \tanh^2(\,\mathbf{W}_{ic}\mathbf{c}_{t-1})\,\right). \tag{2.15}$$

Now, let's consider what happens as t grows: we observe that $\mathbf{x}_t$ and $\mathbf{h}_t$ are bounded to a limited interval. In particular, $\mathbf{x}_t$ is a sample of the input data, and $\mathbf{h}_t$ is bounded in the interval $[-1, 1]$ by construction. Instead, the cell $\mathbf{c}_t$ can grow linearly with the number of recursive steps, making its domain extremely task-dependent. This is a well-known problem, which motivated the introduction of the forget gate in the original LSTM structure [61]. Despite this, the range of possible values of c_t cannot be restricted to a fixed domain. The hyperbolic tangent non-linearity helps to avoid an excessive influence of the unbounded cell state in the gate mechanics, hence preventing unwanted saturation. As it can be seen in Eq. 2.7, 2.8, and 2.9, the term related to the cell state is bounded in the interval $[-1, 1]$. Additionally, it helps screen the connection weights $\mathbf{W}_{\star c}$ from unstable updates.

Even if $\mathbf{c}_t$ grew linearly with the number of time steps, its influence on the sigmoid argument would be mitigated, and it could not take the sigmoid function into its saturated regime against the other two terms driven by $\mathbf{x}$ and $\mathbf{h}$ respectively. On the other hand, the growth of the cell state would push the hyperbolic tangent towards its own saturated regime. This behavior helps protect the weight matrix employed in the connection from unstable updates.

Peephole Connections and their Limitations. We now turn our attention to a related connection, namely the peephole connection [60], which is no longer common in the LSTM formulation. Peephole connections were introduced by Gers and Schmidhuber in [60], and enrich the LSTM equations with recurrent weights $\mathbf{W}_{\star c}$, $\star \in \{\mathbf{i}, \mathbf{f}, \mathbf{o}\}$:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i) \tag{2.16}$$
$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f) \tag{2.17}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o), \tag{2.18}$$

with $\mathbf{W}_{\star c}$ generally constrained to be diagonal [67, 69]. While this formulation allows for a more precise control of the gates, there are two issues that limit its effectiveness. In this case, the local gradient at time t is expressed by:

$$\frac{\partial \mathbf{i}_t}{\partial \bar{\mathbf{i}}_t} = \frac{\partial}{\partial \bar{\mathbf{i}}_t}\sigma\left(\bar{\mathbf{i}}_t\right) = diag\left[\sigma\left(\bar{\mathbf{i}}_t\right) \odot \left(\mathbf{1} - \sigma\left(\bar{\mathbf{i}}_t\right)\right)\right], \tag{2.19}$$

with $\bar{\mathbf{i}}_t$ being the argument of the sigmoid function in Eq 2.16:

$$\bar{\mathbf{i}}_t = \mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i. \tag{2.20}$$

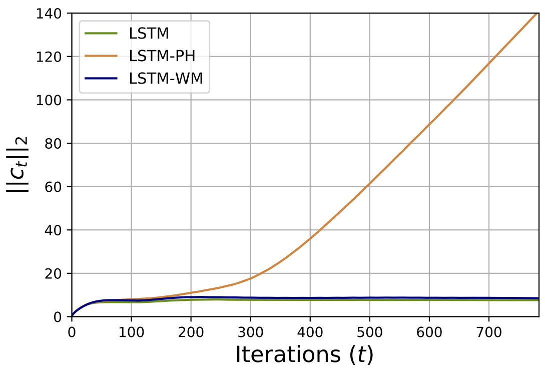


Figure 2.2: The cell state c_t may grow linearly with the number of time steps. Peephole connections directly expose c_t, creating a key issue. Data for this plot is taken from the first training iterations of the sequential MNIST (see § 2.4.2).

In light of this difference, Eq. 2.14 and 2.15 become:

$$\frac{\partial \mathbf{i}_t}{\partial \mathbf{W}_{ic}} = \frac{\partial \mathbf{i}_t}{\partial \bar{\mathbf{i}}_t} \otimes \mathbf{c}_{t-1}. \tag{2.21}$$

We observe that, both in Eq. 2.7 and in Eq 2.16, the magnitude of the product $\mathbf{W}_{ic}\mathbf{c}_{t-1}$ can in principle grow unbounded. The activation function introduced in WMCs squashes this term into a closed bounded interval. In peephole connections, hovewer, this term is added inside the gate without an adequate protection (see Fig. 2.1). The result is that, in the peephole formulation, the sigmoid function applied immediately after could be pushed towards its saturating regime independently from the value of $\mathbf{x}_t$ and $\mathbf{h}_t$. In theory, the LSTM block can recover from this situation by setting all the weights in the peephole connection to 0, but in practice this might not happen if the sigmoid gate is saturated most of the time. Even if the two other summands can compensate for the growth of $\mathbf{W}_{\star c}\mathbf{c}$, hence letting gradients flow through the gate, there is still a key issue that hinders learning. In fact, as shown in Eq. 2.21, the gradients on the recurrent peephole weights grow linearly with $\mathbf{c}$, making updates unstable.

To exemplify this behavior, we report the Euclidean norm of $\mathbf{c}_t$ during the early training stages in Fig. 2.2. After a small number of time steps, the content of the cell floods the gates of the peephole LSTM. A possible consequence would be that both the input and the forget gates would saturate towards 1. In our example, this aspect leads to an additional and uncontrolled growth of the magnitude of $\mathbf{c}_t$. As it can be seen, Working Memory Connections exhibit a much more regular

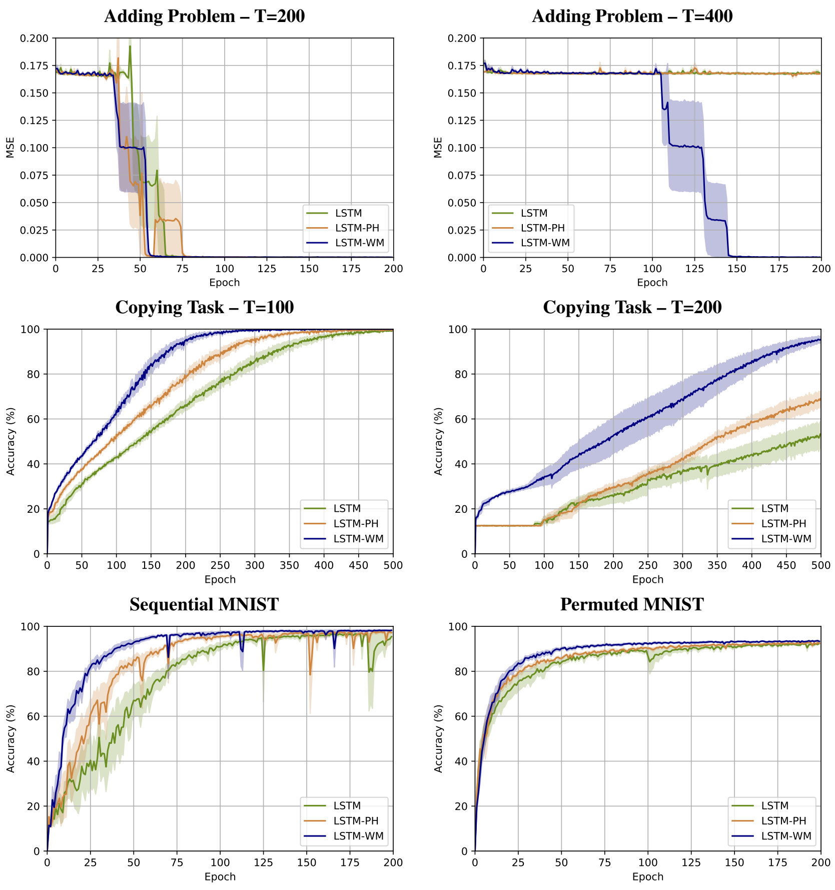


Figure 2.3: Comparison among traditional LSTM, the proposed LSTM with working memory connections, and peephole LSTM. We investigate three different tasks: the adding problem (top), the copying task (center), and the sequential/permuted MNIST (bottom). In all the plots, shading indicates the standard error of the mean.

behavior than peepholes and can prevent the uncontrolled growth of the memory cell.

2.4 Experiments and Results

The effectiveness of Working Memory Connections and their general benefits can be appreciated in many different tasks. The proposed experiments cover a wide area of applications: two different toy problems, digit recognition, language modeling, and image captioning. While the analysis on simple tasks helps to clarify the inherent advantages of the proposed approach, results on more challenging real-world applications motivate a wider adoption of our novel connections, especially for long sequences. We compare our model (LSTM-WC) to a traditional LSTM and to an LSTM with peephole connections (LSTM-PH).

2.4.1 Adding Problem and Copying Tasks

In the adding problem [78], the input to the network consists of a series of T pairs (n_t, f_t), with $0 \leq t < T$. The first element n_t is a real-valued number between 0 and 1, and f_t is a corresponding marker. In the entire sequence, only two markers f_i and f_j are set to 1, while the others are set to 0. The goal is to predict the sum of the corresponding real-valued items $n_i + n_j$, for which $f = 1$. In our experiments, we test with $T = 200$ and $T = 400$, and we measure the performance using mean squared error. For this experiment, the networks have hidden size $N = 128$ and train for 200 epochs. We optimize the parameters using SGD with Nesterov update rule. The learning rate is 10^{-2} (momentum factor 0.9) and the batch size is 128. We also clip the gradient norm to 1.0. Results are reported in Fig. 2.3 (top), where we plot the MSE on the test set for every epoch of training. LSTM-WM achieves the best convergence time for $T = 200$, while the final performance on this setup is similar among the three models. The effectiveness of WMCs is striking in the $T = 400$ setup. In fact, the proposed model solves the adding problem around epoch 145, while the other two architectures cannot learn the task and are stuck on the trivial solution.

In the copying task [78], the network observes a sequence of 10 input symbols, waits for T time steps (we use $T = 100$ and $T = 200$), and then must reproduce the same sequence as output. For this experiment, we adopt the same setup described in [10]. We keep the same implementation details described for the adding problem, except that we train for 500 epochs. In Fig. 2.3 (center), we plot the test accuracy achieved by the three models at each epoch. In both setups, WMCs play an important role in terms of final performance and convergence time. As in the adding problem, the performance gain given by the proposed architecture is more evident when working on longer sequences: for $T = 200$, WMCs outperform peephole LSTM and vanilla LSTM by around $+25\%$ and $+40\%$.

2.4.2 Permuted Sequential MNIST

The sequential MNIST (sMNIST) [97] is the sequential version of the MNIST digit recognition task [98]. In this task, the image pixels are fed sequentially to the network (from left to right, and top to bottom). The permuted sequential MNIST

Model	**sMNIST**	**pMNIST**
iRNN [97]	97.00	82.00
uRNN [10]	95.10	91.40
h-detach [12]	98.50	92.30
LSTM ($h = 128$)	98.16	92.94
LSTM ($h = 256$)	97.68	**93.97**
LSTM-PH ($h = 128$)	98.58	93.25
LSTM-PH ($h = 256$)	98.33	93.40
LSTM-WM ($h = 128$)	**98.63**	**93.97**

Table 2.1: Test accuracy on the sequential MNIST task.

(pMNIST) is a sequential version of the MNIST digit recognition problem in which the pixels are permuted in a random but fixed order. In both tasks, the goal is to predict the correct digit label after the last input pixel. Following the setup proposed in [12], we use 50k images for training, 10k for validation, and 10k to test our models. The experimental setup is as follows. We set the hidden size to $N = 128$ for all the networks, and train for 200 epochs using SGD with learning rate 10^{-2} and batch size 128 (momentum 0.9 and Nesterov update rule). We clip the gradient norms to 1.0.

Fig. 2.3 (bottom) reports the mean test accuracy of the three LSTM variants for both setups. We report the standard error of the mean as a shaded area. For the sMNIST task, peephole LSTM performs slightly better than vanilla LSTM. LSTM with Working Memory Connections, instead, outperforms the competing architectures in terms of final accuracy and convergence speed. In particular, our architecture employs only 50 epochs to get above 92% accuracy, while other models are still generally stuck around 65% (vanilla LSTM) and 82% (LSTM-PH). In this experiment, we also find out that WMCs help stabilize training. In fact, the area given by the standard error of the mean is much thicker for our approach than for the other two variants, in particular during the early stages of training. On the pMNIST task, all the models achieve good final results, with LSTM with Working Memory Connections still being the best option.

Numerical results, reported in Table 2.1, confirm that our model outperforms the classic LSTM by a discrete margin (+0.47% and +1.03% on the sequential and permuted MNIST respectively). Since WMCs introduce additional learnable parameters in the LSTM structure, we also compare with vanilla and peephole LSTM with increased hidden size (256 instead of 128). Note that, in this setting, LSTM and LSTM-PH have more than 2× the number of learnable parameters of LSTM-WM. Despite this, LSTM-WC achieves the best results on both tasks. It is worth noting that, while additional parameters in vanilla LSTM improves the results on pMNIST, they are not helpful in the sMNIST task. The flexibility given by WMCs, instead, allows the proposed model to achieve the best result in

	Test Bit per Character (BPC)			
	Fixed # Params ($\sim 2.2M$)		Fixed # Hidden Units (512)	
Model	$T_{PTB} = 150$	$T_{PTB} = 300$	$T_{PTB} = 150$	$T_{PTB} = 300$
LSTM	1.334 ± 0.0006	1.343 ± 0.0004	1.386 ± 0.0005	1.395 ± 0.0005
LSTM-PH	1.339 ± 0.0048	1.343 ± 0.0009	1.383 ± 0.0004	1.394 ± 0.0005
LSTM-WM	**1.299 ± 0.0005**	**1.302 ± 0.0008**	**1.299 ± 0.0005**	**1.302 ± 0.0008**

Table 2.2: Mean test bit per character on the PTB test set. Error range indicates the standard error of the mean.

both setups. Always in Table 2.1, we compare with two state-of-the-art RNNs [10, 97], and with a training algorithm for LSTM [12]. The proposed LSTM-WC outperforms the competitors in terms of test accuracy.

2.4.3 Penn Treebank Character-Level Language Modeling

Character-level language modeling requires to predict a single character at each time step given an observed sequence of text. In our experiments on the Penn Treebank (PTB) dataset [113], we evaluate the performance of the three different LSTM variants in terms of test mean bits per character (BPC), where lower BPC denotes better performance. We report the results in Table 2.2, where we compare truncated back-propagation through time (T_{PTB}) over 150 and 300 steps. Since our connection introduces new learnable weights, we consider an additional setup in which we keep a fixed number of parameters for the three networks. For this experiment, we follow the setup proposed by Merity *et al.* [115], with the only exception that we employ a single LSTM layer instead of three. The advantage of using Working Memory Connections is more evident for equal number of hidden units, where the proposed architecture overcomes the vanilla LSTM and peephole LSTM by a significant margin. Even when the number of parameters is fixed for all the models, LSTM-WC outperforms the competitors by 0.035 and 0.041 BPC for $T_{PTB} = 150$ and $T_{PTB} = 300$ respectively. It is worth noting that peephole LSTM performs similarly to or even worse than vanilla LSTM on this task.

2.4.4 Image Captioning

We evaluate the performance of our LSTM with Working Memory Connections on the image captioning task, which consists of generating textual descriptions for images. We apply our approach to two different captioning models: Show and Tell [166] and Up-Down [5]. The first model includes a single LSTM layer and does not employ attention, while the second is composed of two LSTM layers and integrates attention mechanisms over image regions. We use the Microsoft COCO dataset [103] following the splits defined in [87]. To represent images, we employ

Model	**BLEU-1**	**BLEU-4**	**METEOR**	**ROUGE**	**CIDEr**	**SPICE**
No Attention, ResNet-152						
LSTM	70.9	27.9	24.4	51.7	92.0	17.6
GRU	69.5	26.2	22.7	50.4	82.3	15.6
LSTM-PH	**71.4**	27.8	24.3	51.7	91.1	17.5
LSTM-WM	**71.4**	**28.3**	**24.6**	**52.4**	**94.0**	**17.8**
Attention, Faster R-CNN						
LSTM	75.9	**36.1**	27.4	56.3	111.9	20.3
GRU	76.0	**36.1**	27.0	56.5	111.0	20.2
LSTM-PH	75.8	35.9	27.3	56.3	111.5	20.2
LSTM-WM	**76.2**	**36.1**	**27.5**	**56.5**	**112.7**	**20.4**

Table 2.3: Image captioning results on COCO test set.

a global feature vector extracted from the average pooling layer of ResNet-152 [73] for the Show and Tell model, and multiple feature vectors extracted from Faster R-CNN [137] for the Up-Down architecture. We train both models with Adam optimizer [90] using a learning rate equal to 10^{-4}. All other hyper-parameters are left the same as those suggested in the original papers.

Numerical results are reported in Table 2.3 using standard captioning evaluation metrics (*i.e.* BLEU-1, BLEU-4 [126], METEOR [14], ROUGE [102], CIDEr [165], and SPICE [4]). For all these, higher metric results indicate better performance, with CIDEr being the metric that best correlates with human judgment. In both settings, our LSTM-WM outperforms traditional LSTM and LSTM-PH by a clear margin. Specifically, LSTM-WM improves the vanilla LSTM results by 2.0 CIDEr points on the model without attention and 0.8 CIDEr points on the model with attention over image regions, demonstrating the contribution of WMCs also for this task. As an additional comparison, we replace the LSTM layers with GRU layers. Numerical results suggest that there is not a clear advantage in using GRUs instead of LSTMs for this task. In Fig. 2.4, we plot the metric gap between LSTM-WM and the two competitors in terms of METEOR and CIDEr. On the X-axis we report the length of the generated captions, meaning that we consider the first x words of each predicted sentence. On the Y-axis, a 0 value means that our proposal performs equally, *i.e.* has no performance gap *w.r.t.* the competitor, while a higher value indicates better performance for our model. With this analysis, we aim to check whether the improvement given by WMCs can be restricted to a particular subset of the dataset. As one can observe, the metric gap generally increases with the caption length, especially *w.r.t.* peephole LSTM. We can deduce that the contribution of WMCs escalates with the number of time steps.

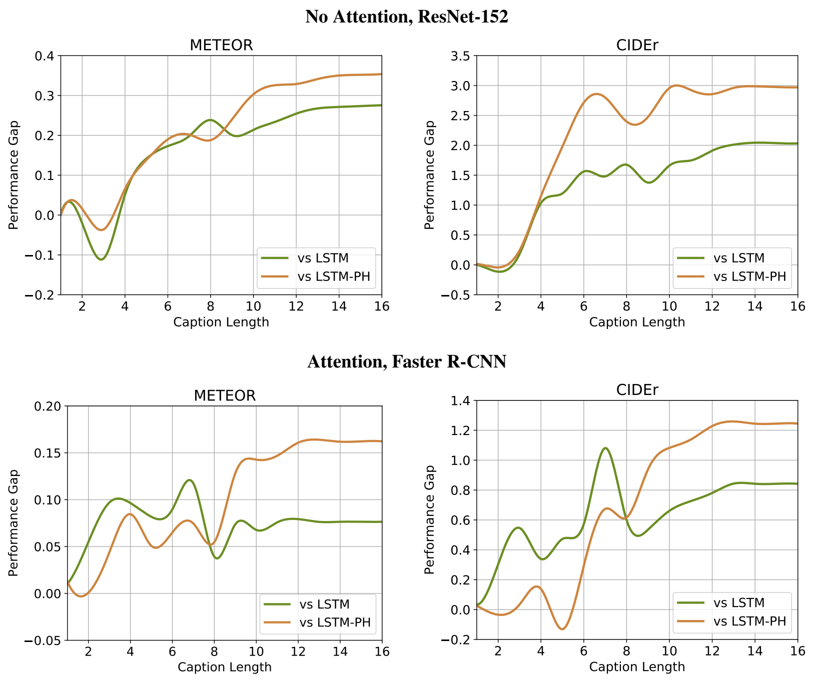


Figure 2.4: Metric gaps on the image captioning task for increasing instruction lengths.

2.5 Discussion

With Working Memory Connections, we show that information stored in the LSTM cell should be accessible in the gate structure. We compare the performance of WMCs to a similar approach named peephole connections [60], and to vanilla LSTM. We find out that the structure of WMCs allows for two distinct improvements:

1. *A more precise control of the gates*. The multiplicative gates in the LSTM block must regulate the information flowing through the cell, but they cannot access the state of that same cell in the traditional LSTM formulation. The presence of the cell state in the multiplicative gates motivates the improvements of LSTM-WM *w.r.t.* vanilla LSTM.

2. *Increased stability during training* compared to peephole connections. Exposing different projections of the cell state without squashing its content seems to be a critical point for the LSTM-PH. This element of novelty in

our design explains why WMCs provide a boost in performance even when peepholes fail.

As a consequence of these two improvements, WMCs incorporate the theoretical benefits of peephole connections, originally described by Gers *et al.* [60], with the training stability and versatility of vanilla LSTM.

It is worth noting that, for tasks that do not require to access the content of the memory cell, Working Memory Connections would not probably bring any benefit in the LSTM formulation, while peepholes might still hinder the whole learning process because of unstable updates.

At the same time, when training stacked LSTMs, the benefits given by WMCs may become less significant. We suppose that this is due to the increased complexity in the network structure, where multiple LSTM blocks can interact through the various layers. Similarly, many architectures employ LSTMs as building blocks together with different components, and the influence of WMCs in these compound deep networks cannot be easily determined. Experiments on image captioning, proposed in this Chapter, partially answer this question and prove that WMCs afford a small yet existing improvement even in this scenario.

2.6 Conclusion

A current limitation of Long Short-Term Memory Networks consists in not letting the cell state influence the gate dynamics directly. In this Chapter, we propose Working Memory Connections (WMCs) for LSTM, which provide an efficient way of using intra-cell knowledge inside the network. The proposed design performs noticeably better than the vanilla LSTM and overcomes important issues in previous formulations. We formally motivate this improvement as a consequence of more stable training dynamics. Experimental results reflect the theoretical benefits of the proposed approach and motivate further study in this direction.

Chapter 3

Explore and Explain: Joining Exploration and Recounting

Embodied AI has been recently gaining attention as it aims to foster the development of autonomous and intelligent agents. In this Chapter, we devise a novel embodied setting in which an agent needs to explore a previously unknown environment while recounting what it sees during the path. In this context, the agent needs to navigate the environment driven by an exploration goal, select proper moments for description, and output natural language descriptions of relevant objects and scenes. Our model integrates a novel self-supervised exploration module with penalty, and a fully-attentive captioning model for explanation. Also, we investigate different policies for selecting proper moments for explanation, driven by information coming from both the environment and the navigation. Experiments are conducted on photorealistic environments from the Matterport3D dataset and investigate the navigation and explanation capabilities of the agent as well as the role of their interactions.

This Chapter is related to publication [3], as reported in the List of Publications.

3.1 Introduction

Only a few decades ago, intelligent robots that could autonomously walk and talk existed only in the bright minds of book and movie authors. People used to think about artificial intelligence only as a fictional feature, as the machines they interacted with were purely reactive and showed no form of autonomy. Nowadays, intelligent systems are everywhere, with deep learning being the main engine of the so-called AI revolution. More recently, advances in the field of Embodied AI aim to foster the next generation of autonomous and intelligent robots. Progress in this field includes visual navigation and instruction following [6], even though current research is also focused on the creation of new research platforms for simulation and evaluation [145, 175]. At the same time, tasks at the intersection of computer vision and natural language processing are of particular interest for the community, with image captioning being one of the most active areas [5, 42, 87]. By describing the content of an image or a video, captioning models can bridge the gap between the black-box architecture and the user.

In this Chapter, we propose a new task at the intersection of Embodied AI, computer vision, and natural language processing, and aim to create a robot that can navigate through a new environment and describe what it sees. We call this new task *Explore and Explain* since it tackles the problem of joint exploration and captioning (Fig. 3.1). In this schema, the agent needs to perceive the environment around itself, navigate it driven by an exploratory goal, and describe salient objects and scenes in natural language. Beyond navigating the environment and translating visual cues in natural language, the agent also needs to identify appropriate moments to perform the explanation step.

It is worthwhile to mention that both exploration and explanation feature significant challenges. Effective exploration without any previous knowledge of the environment cannot exploit a reference trajectory and the agent cannot be trained with classic methods from reinforcement learning [171]. To overcome this problem, we design a self-supervised exploration module that is driven solely by curiosity towards the new environment. In this setting, rewards are more sparse than in traditional setups and encourage the agent to explore new places and to interact with the environment.

While we are motivated by recent works incorporating curiosity in Atari and other exploration games [2, 28, 129], the effectiveness of a curiosity-based approach in a photorealistic, indoor environment has not been tested extensively. Some preliminary studies [136] suggest that curiosity struggles with embodied exploration. In this Chapter, we show that a simple modification of the reward function can lead to striking improvements in the exploration of unseen environments. Additionally, we encourage the agent to produce a description of what it sees throughout the navigation. In this way, we match the agent internal state (the measure of curiosity) with the variety and the relevance of the generated captions.

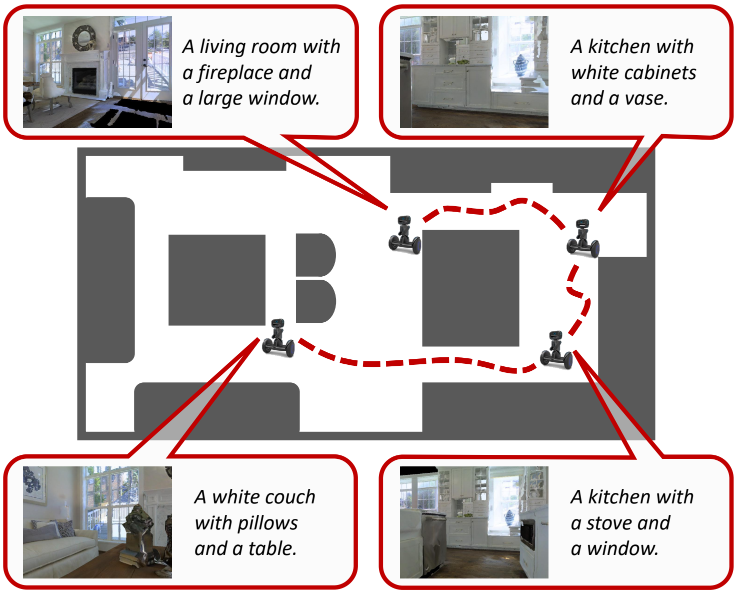

A living room with a fireplace and a large window.
A kitchen with white cabinets and a vase.
A white couch with pillows and a table.
A kitchen with a stove and a window.

Figure 3.1: We propose a novel setting in which an embodied agent performs joint curiosity-driven exploration and explanation in unseen environments. While navigating the environment, the agent must produce informative descriptions of what it sees, providing a means of interpreting its internal state.

Such matching offers a proxy for the desirable by-product of interpretability. In fact, by looking at the caption produced, the user can more easily interpret the navigation and perception capabilities of the agent, and the motivations of the actions it takes [41]. In this sense, our work is related to goal-driven explainable AI, *i.e.* the ability of autonomous agents to explain their actions and the reasons leading to their decisions [8].

Previous work on image captioning has mainly focused on recurrent neural networks. However, the rise of Transformer [164] and the great effectiveness shown by the use of self-attention have motivated a shift towards recurrent-free architectures. Our captioning algorithm builds upon recent findings on the importance of fully-attentive networks for image captioning and incorporates self-attention both during the encoding of the image features and in the decoding phase.

Finally, to bridge exploration and recounting, our model can count on a novel speaker policy, which regulates the speaking rate of our captioner using information coming from the agent perception. We call our architecture eX^2, from the name of the task: ***Ex**plore and **Ex**plain*.

Our main contributions are as follows:

- We propose a new setting for Embodied AI, *Explore and Explain* in which the agent must jointly deal with two challenging tasks: exploration and captioning of unseen environments.
- We devise a novel solution involving curiosity for exploration. Thanks to curiosity, we can learn an efficient policy which can easily generalize to unseen environments.
- We are the first, to the best of our knowledge, to apply a captioning algorithm exclusively to indoor environment for robotic exploration. Results are encouraging and motivate further research.

3.2 Related Work

Our work is related to the literature on embodied visual exploration, curiosity-driven exploration, and captioning. In the following, we provide an overview of the most important work in these settings, and we briefly describe the most commonly used interactive environments for navigation agents.

Embodied Visual Exploration. Current research on Embodied AI is mainly focused on tasks that require navigating indoor locations. Vision-and-language navigation [6, 95], point-goal and object-goal navigation [3, 171, 185] are all tasks involving the ability for the agent to move across a previously unknown environment. Very recently, Ramakrishnan *et al.* [136] highlighted the importance of visual exploration in order to pre-train a generic embodied agent. While their study is mainly focused on exploration as a mean to gather information and to prepare for future tasks, we investigate the role of surprisal for exploration and the consistency between navigation paths and the descriptions given by the agent during the episodes.

Curiosity-driven Exploration. Curiosity-driven exploration is an important topic in reinforcement learning literature. In this context, [125] provides a good summary on early works on intrinsic motivation. Among them, Schmidhuber [146] and Sun *et al.* [157] proposed to use information gain and compression as intrinsic rewards, while Klyubin *et al.* [92], and Mohamed and Rezende [116] adopted the concept of empowerment as reward during training. Differently, Houthooft *et al.* [81] presented an exploration strategy based on the maximization of information gain about the agent's belief of environment dynamics. Another common approach for exploration is that of using state visitation counts as intrinsic rewards [17, 160]. Our work follows the strategy of jointly training forward and backward models for learning a feature space, which has demonstrated to be effective in curiosity-driven exploration in Atari and other exploration games [2, 28, 129]. To the best of our knowledge, we are the first to investigate this type of exploration algorithms in photorealistic indoor environments.

Interactive Environments. When it comes to the training of intelligent agents, an important role is played by the underlying environment. A first test bed for research in reinforcement learning has been provided by the Atari games [18, 27]. However, these kind of settings are not suitable for navigation and exploration in general. To solve this problem, many maze-like environments have been proposed [16, 89]. However, agents trained on synthetic environments hardly adapt to real world scenarios, because of the drastic change in terms of appearances. Simulating platforms like Habitat [145], Gibson [175], and Matterport3D simulator [6] provide a photorealistic environment to train navigation agents. Some of these simulators only provide RGB equirectangular images as visual input [6], while others employ the full 3D model and implement physic interactions with the environment [145, 175].

Automatic Captioning. In the last few years, a large number of models has been proposed for image captioning [5, 139, 177]. The majority of them use recurrent neural networks as language models and a representation of the image which might be given by the output of a CNN [139, 167], or by a time-varying vector extracted with attention mechanisms over either a spatial grid of CNN features [177] or multiple image region vectors extracted from a pre-trained object detector [5]. Regarding the training strategies, notable advances have been made by using reinforcement learning to optimize non-differentiable captioning metrics [139]. Recently, following the strong advent of fully-attentive mechanisms in sequence modeling tasks [164], different Transformer-based captioning models have been presented [42, 74].

3.3 Proposed Method

The proposed method consists of three main parts: a navigation module, a speaker policy, and a captioner. The last two components constitute the speaker module, which is used to explain the agent first-person point of view. The explanation is elicited by our speaker module basing on the information gathered during the navigation. Our architecture is depicted in Fig. 3.2 and detailed below.

3.3.1 Navigation Module

The navigation policy takes care of the agent displacement inside the environment. At each time step t the agent acquires an observation x_t from the surroundings, performs an action a_t, and gets the consequent observation x_{t+1}. The moves available to the agent are simple, atomic actions such as *rotate 15 degrees* and *step ahead*.

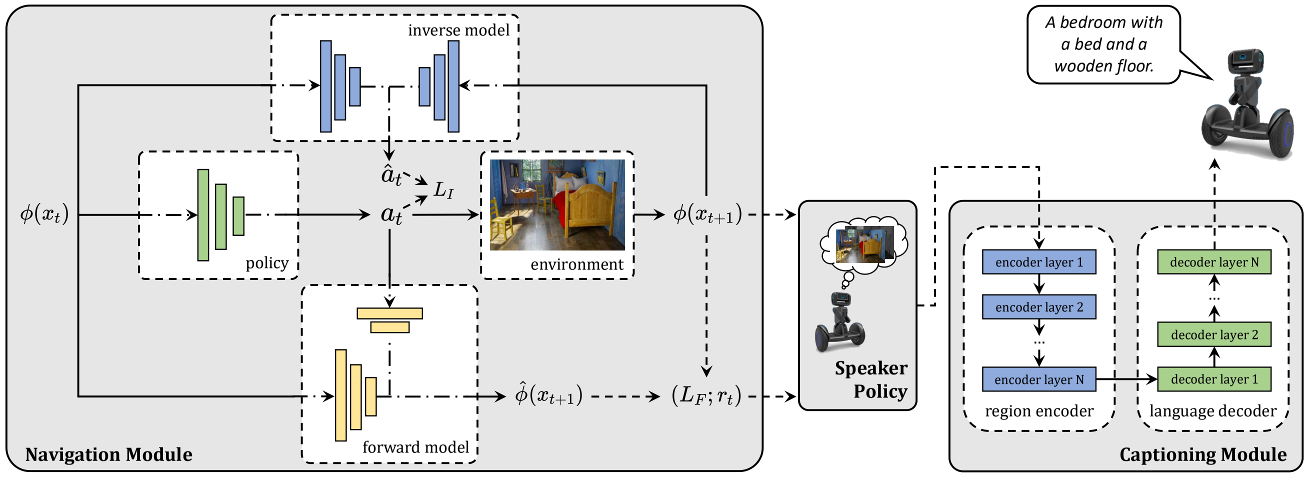


Figure 3.2: Overview of our eX^2 framework for navigation and captioning. Our model is composed of three main components: a navigation module which is in charge of exploring the environment, a captioning module which produces a textual sentence describing the agent point of view, and a speaker policy that connects the previous modules and activates the captioning component based on the information collected during the navigation.

Our navigation module consists of three main components: a feature embedding network, a forward model, and an inverse model. The discrepancy of the predictions of dynamics models with the actual observation is measured by a reward signal r_t, which is then used to stimulate the agent to move towards more informative states.

Embedding Network. At each time step t, the agent observes the environment and gathers x_t. This observation corresponds to the raw RGB-D pixels coming from the forward-facing camera of the agent. Yet, raw pixels are not optimal to encode the visual information [28]. For this reason, we employ a convolutional neural network ϕ to encode a more efficient and compact representation of the surrounding environment. We call this embedded representation $\phi(x_t)$. To ensure that the features observed by the agent are stable throughout the training, we do not change the set of parameters θ_ϕ during training. This approach is shown to be efficient for generic curiosity-based agents [28].

Forward Dynamics Model. Given an agent with policy $\pi(\phi(x_t); \theta_\pi)$, represented by a neural network with parameters θ_π, the selected action at timestep t is given by:

$$a_t \sim \pi\Big(\phi(x_t); \theta_\pi\Big). \tag{3.1}$$

After executing the chosen action, the agent observes a new visual stimulus $\phi(x_{t+1})$. The problem of predicting the next observation given the current input and action to be performed can be defined as a forward dynamics problem:

$$\hat{\phi}(x_{t+1}) = f\Big(\phi(x_t), a_t; \theta_F\Big), \tag{3.2}$$

where $\hat{\phi}(x_{t+1})$ is the predicted visual embedding for the next observation x_{t+1} and f is the forward dynamics model with parameters θ_F. The forward model is trained to minimize the following loss function:

$$L_F = \frac{1}{2}\left\|\hat{\phi}(x_{t+1}) - \phi(x_{t+1})\right\|_2^2 \tag{3.3}$$

Inverse Dynamics Model. Given two consecutive observations (x_t, x_{t+1}), the inverse dynamics model aims to predict the action performed at timestep t:

$$\hat{a}_t = g\Big(\phi(x_t), \phi(x_{t+1}); \theta_I\Big), \tag{3.4}$$

where $\hat{a}_t$ is the predicted estimate for the action a_t and g is the inverse dynamics model with parameters θ_I. In our work, the inverse model g predicts a probability distribution over the possible actions and it is optimized to minimize the cross-entropy loss with the ground-truth action a_t performed in the previous time step:

$$L_I = y_t \log \hat{a}_t, \tag{3.5}$$

where y_t is the one-hot representation for a_t.

Curiosity-driven Exploration. The agent exploration policy $\pi(\phi(x_t);\theta_\pi)$ is trained to maximize the expected sum of rewards:

$$\max_{\theta_\pi} \mathbb{E}_{\pi(\phi(x_t);\theta_\pi)}\left[\sum_t r_t\right], \tag{3.6}$$

where the exploration reward r_t at timestep t, also called surprisal [1], is given by our forward dynamics model:

$$r_t = \frac{\eta}{2}\left\|f\big(\phi(x_t), a_t\big) - \phi(x_{t+1})\right\|_2^2, \tag{3.7}$$

with η being a scaling factor. The overall optimization problem can be written as a composition of Eq. 3.3, 3.5, and 3.6:

$$\min_{\theta_\pi,\theta_F,\theta_I}\left[-\lambda\mathbb{E}_{\pi(\phi(x_t);\theta_\pi)}\Big[\Sigma_t r_t\Big] + \beta L_F + (1-\beta)L_I\right] \tag{3.8}$$

where λ weighs the importance of the intrinsic reward signal *w.r.t.* the policy loss, and β balances the contributions of the forward and inverse models.

Penalty for Repeated Actions. To encourage diversity in our policy, we devise a penalty which triggers after the agent has performed the same move for $\tilde{t}$ timesteps. This prevents the agent from always picking the same action and encourages the exploration of different combinations of atomic actions.

We can thus rewrite the surprisal in Eq. 3.7 as:

$$r_t = \frac{\eta}{2}\left\|f\big(\phi(x_t), a_t\big) - \phi(x_{t+1})\right\|_2^2 - p_t, \tag{3.9}$$

where p_t is the penalty at time step t. In the simplest formulation, p_t can be modeled with a scalar which is either 0 or equal to a constant $\tilde{p}$, after an action has been repeated $\tilde{t}$ times.

3.3.2 Speaker Policy

As the navigation proceeds, new observations x_t are acquired and rewards r_t are obtained at each time step. Based on these, a speaker policy can be defined, that activates the captioning module. Different types of information from the environment and the navigation module allow defining different policies. In this work, we consider three policies, namely: object-driven, depth-driven, and curiosity-driven.

Object-driven Policy. Given the RGB component of the observation x_t, relevant objects can be recognized. When at least a minimum number **O** of such objects are observed, the speaker policy triggers the captioner. The idea behind this policy is to let the captioner describe the scene only when objects that allow connoting the different views are present.

Depth-driven Policy. Given the depth component of the observation x_t, the speaker policy activates the captioner when the mean depth value perceived $\mathbf{D}$ is above a certain threshold. This way, the captioner is triggered only depending on the distance of the agent from generic objects, regardless of their semantic category.

Curiosity-driven Policy. Given the surprisal reward defined as in Eq. 3.7 and possibly cumulated over multiple timesteps, $\mathbf{S}$, the speaker policy triggers the captioner when $\mathbf{S}$ is above a certain threshold. This policy is independent of the type of information perceived from the environment but is instead closely related to the navigation module. Thus, it helps to match the agent's internal state with the generated captions more explicitly than the other policies.

3.3.3 Captioning Module

When the speaker policy activates, a captioning module is in charge of producing a description in natural language given the current observation x_t. Following recent literature on the topic, we here employ a visual encoder based on image regions [138], and a decoder which models the probability of generating one word given previously generated ones. In contrast to previous captioning approaches based on recurrent networks, we propose to use a fully-attentive model for both the encoding and the decoding stage, building on the Transformer model [164].

Region Encoder. Given a set of features from image regions $R = \{r_1, ..., r_N\}$ extracted from the agent visual view, our encoder applies a stack of self-attentive and linear projection operations. As the former can be seen as convolutions on a graph, the role of the encoder can also be interpreted as that of learning visual relationships between image regions. The self-attention operator S builds upon three linear projections of the input set, which are treated as queries, keys and values for an attention distribution. Stacking region features R in matrix form, the operator can be defined as follows:

$$S(R) = \mathsf{Attention}(W_q R, W_k R, W_v R), \tag{3.10}$$

$$\mathsf{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V. \tag{3.11}$$

The output of the self-attention operator is a new set of elements $S(R)$, with the same cardinality as R, in which each element of R is replaced with a weighted sum of the values, *i.e.* of linear projections of the input.

Following the structure of the Transformer model, the self-attention operator S is followed by a position-wise feed-forward layer, and each of these two operators is encapsulated within a residual connection and a layer norm operation. Multiple layers of this kind are then applied in a stack fashion to obtain the final encoder.

Language Decoder. The output of the encoder module is a set of region encodings $\tilde{R}$ with the same cardinality of R. We employ a fully-attentive decoder which is conditioned on both previously generated words and region encodings, and is in charge of generating the next tokens of the output caption. The structure of our decoder follows that of the Transformer [164], and thus relies on self-attentive and cross-attentive operations.

Given a partially decoded sequence of words $W = \{w_0, w_1, ..., w_\tau\}$, each represented as a one-hot vector, the decoder applies a self-attention operation in which W is used to build queries, keys and values. To ensure the causality of this sequence encoding process, we purposely mask the attention operator so that each word can only be conditioned to its left-hand sub-sequence, *i.e.* word w_t is conditioned on $\{w_{t'}\}_{t' \leq t}$ only. Afterwards, a cross-attention operator is applied between W and $\tilde{R}$ to condition words on regions, as follows:

$$C(W, \tilde{R}) = \mathsf{Attention}(W_q W, W_k \tilde{R}, W_v \tilde{R}). \tag{3.12}$$

As in the Transformer model, after a self-attention and a cross-attention stage, a position-wise feed-forward layer is applied, and each of these operators is encapsulated within a residual connection and a layer norm operation. Finally, our decoder stacks together multiple decoder layers, helping to refine the understanding of the textual input.

Overall, the decoder takes as input word vectors, and the t-th element of its output sequence encodes the prediction of a word at time $t + 1$, conditioned on $\{w_t\}_{\leq t}$. After taking a linear projection and a softmax operation, this encodes a probability over words in the dictionary. During training, the model is trained to predict the next token given previous ground-truth words; during decoding, we iteratively sample a predicted word from the output distribution and feed it back to the model to decode the next one, until the end of the sequence is reached. Following the usual practice in image captioning literature, the model is trained to predict an end-of-sequence token to signal the end of the caption.

3.4 Experimental Setup

3.4.1 Dataset

The main testbed for this work is Matterport3D [31], a photorealistic dataset of indoor environments. Some of the buildings in the dataset contain outdoor components like swimming pools or gardens, raising the difficulty of the exploration task. The dataset is split into 61 scenes for training, 11 for validation, and 18 for testing. It also provides instance segmentation annotations that we use to evaluate the captioning module. Overall, the dataset is annotated with 40 different semantic categories. For both training and testing, we use the episodes provided by Habitat API [145] for the PointGoal navigation task, employing only the starting point of each episode. The size of the training set amounts to a total of 5M episodes, while the test set is composed of 1 008 episodes.

3.4.2 Evaluation Protocol

Navigation Module. To quantitatively evaluate the navigation module, we use a curiosity-based metric: we extract the sum of the surprisal values defined in Eq. 3.7 every 20 steps performed by the agent, and then we compute the average over the number of test episodes.

Captioning Module. Standard captioning methods are usually evaluated by comparing each generated caption against the corresponding ground-truth sentences. However, in this setting, only the information on which objects are present on the scene is available, thanks to the semantic annotations provided by the Matterport3D dataset. Therefore, to evaluate the performance of our captioning module, we define two different metrics: a soft coverage measure that assesses how the predicted caption covers all the ground-truth objects, and a diversity score that measures the diversity in terms of described objects of two consecutively generated captions.

In details, for each caption generated according to the speaker policy, we compute the soft coverage measure between the ground-truth set of semantic categories and the set of nouns in the caption. Given a predicted caption, we firstly extract all nouns $\boldsymbol{n}$ from the sentence and we compute the optimal assignment between them and the set of ground-truth categories $\boldsymbol{c}^*$, using distances between word vectors and the Hungarian algorithm [94]. We then define an intersection score between the two sets as the sum of assignment profits. Our coverage measure is computed as the ratio of the intersection score and the number of ground-truth semantic classes:

$$\mathsf{Cov}(\boldsymbol{n}, \boldsymbol{c}^*) = \frac{\mathrm{I}(\boldsymbol{n}, \boldsymbol{c}^*)}{\#\boldsymbol{c}^*}, \tag{3.13}$$

where $\mathrm{I}(\cdot, \cdot)$ is the intersection score, and the $\#$ operator represents the cardinality of the set of ground-truth categories.

Since images may contain small objects which not necessarily should be mentioned in a caption describing the overall scene, we define a variant of the coverage measure by thresholding over the minimum object area. In this case, we consider $\boldsymbol{c}^*$ as the set of objects whose overall areas are greater than the threshold.

For the diversity measure, we consider the sets of nouns extracted from two consecutively generated captions, indicated as $\boldsymbol{n}_t$ and $\boldsymbol{n}_{t+1}$, and we define a soft intersection over union score between the two sets of nouns. Also in this case, we compute the intersection score between the two sets using word distances and the Hungarian algorithm to find the optimal assignment. Recalling that set union can be expressed in function of an intersection, the final diversity score is computed by subtracting the intersection over union score from 1 (*i.e.* the Jaccard distance between the two sets):

$$\mathsf{Div}(\boldsymbol{n}_t, \boldsymbol{n}_{t+1}) = 1 - \frac{\mathrm{I}(\boldsymbol{n}_t, \boldsymbol{n}_{t+1})}{\#\boldsymbol{n}_t + \#\boldsymbol{n}_{t+1} - \mathrm{I}(\boldsymbol{n}_t, \boldsymbol{n}_{t+1})}, \tag{3.14}$$

where $\mathrm{I}(\cdot, \cdot)$ is the intersection score previously defined, and the $\#$ operator represents the cardinality of the sets of nouns.

We evaluate the diversity of generated captions with respect to the three speaker policies described in Sec. 3.3.2 and considering different thresholds for each policy (*i.e.* number of objects, mean depth value, and surprisal score). For each speaker policy and selected threshold, the agent is triggered a different number of times thus generating a variable number of captions during the episode. We define the agent's overall loquacity as the number of times it is activated by the speaker policy according to a given threshold. In the experiments, we report the loquacity values averaged over the test episodes.

3.4.3 Implementation and Training Details

Navigation Module. Navigation agents are trained using only visual inputs, with each observation converted to grayscale, cropped and re-scaled to a 84×84 size. A stack of four historical observations $[x_{t-3}, x_{t-2}, x_{t-1}, x_t]$ is used for training in order to model temporal dependencies. We adopt PPO [147] as learning algorithm and employ Adam [90] as optimizer. The learning rate for all networks is set to 10^{-4} and the length of rollouts is equal to 128. For each rollout we make 3 optimization steps. The features $\phi(x_t)$ used by the forward and backward dynamics networks are 512-dimensional and are obtained using a randomly initialized convolutional network ϕ with fixed weights θ_ϕ, following the approach in [28].

The model is trained using the splits described in Sec. 3.4.1, stopping the training after 10 000 updates of the agent. The length of an exploration episode is 1 000 steps. In our experiments, we set the parameters reported in Eq. 3.8 to $\lambda = 0.1$ and $\beta = 0.2$, respectively. Concerning the penalty p_t given to the agent to stimulate diversity (Eq. 3.9), we set $p_t = \tilde{p} = 0.01$ after the same action is repeated for $\tilde{t} = 5$ times.

Speaker Policy. For the object-driven policy, we use the instance segmentation annotations provided by the Matterport3D dataset [31]. For this policy, we select 15 of the 40 semantic categories in the dataset, discarding the contextual ones, which would not be discriminative for the different views acquired by the agent, as for example *wall*, *floor*, and *ceiling*. This way, we can better evaluate the effect of the policy without it being affected by the performance of an underlying object detector of recognizing objects in the agent's current view. Also for the depth-driven policy, we obtain the depth information of the current view from the Matterport3D dataset, averaging the depth values to extract a single score. In the curiosity-driven policy, we consider the sum of surprisal scores extracted over the last 20 steps, obtained by the agent during navigation.

Captioning Module. To represent image regions, we use Faster R-CNN finetuned on the Visual Genome dataset [5, 138], thus obtaining a 2048-dimensional feature vector for each region. To represent words, we use one-hot vectors and linearly project them to the input dimensionality of the model, d. We also employ sinusoidal positional encodings [164] to represent word positions inside the sequence, and sum the two embeddings before the first encoding layer. In both region encoder and language decoder, we set the dimensionality d of each layer to 512, the number of heads to 8, and the dimensionality of the inner feed-forward layer to 2048. We use dropout with keep probability 0.9 after each attention layer and after position-wise feed-forward layers.

Following a standard practice in image captioning [5, 139], we train our model in two phases using image-caption pairs coming from the COCO dataset [103]. Firstly, the model is trained with cross-entropy loss to predict the next token given previous ground-truth words. Then, we further optimize the sequence generation using reinforcement learning employing a variant of the self-critical sequence training [139] on sequences sampled using beam search [5]. Pre-training with cross-entropy loss is done using the learning rate scheduling strategy defined in [164] with a warmup equal to $10\,000$ iterations. Then, during finetuning with reinforcement learning, we use the CIDEr-D score [165] as reward and a fixed learning rate equal to 5^{-6}. We train the model using the Adam optimizer [90] and a batch size of 50. During CIDEr-D optimization and caption decoding, we use beam search with a beam size equal to 5. To compute coverage and diversity metrics and for extracting nouns from predicted captions, we use the spaCy NLP toolkit[1]. We use GloVe word embeddings [131] to compute word similarities between nouns and semantic class names.

3.5 Experimental Results

3.5.1 Navigation Results

As defined in Sec. 3.4.2, we evaluate the performance of our navigation agents by computing the average surprisal score over test episodes. Results are reported

[1]`https://spacy.io/`

Navigation Module	**Surprisal**
Random Exploration	0.333
eX^2 w/o Penalty for repeated actions (RGB only)	0.193
eX^2 w/o Penalty for repeated actions (Depth only)	0.361
eX^2 w/o Penalty for repeated actions (RGB + Depth)	0.439
eX^2	**0.697**

Table 3.1: Surprisal scores for different navigation policies obtained during the agent exploration of the environment.

in Table 3.1 and show that our complete method (eX^2) outperforms all other variants, achieving a significantly greater surprisal score than our method without penalty. In particular, the final performance greatly benefits from using both visual modalities (RGB and depth), instead of using a single visual modality to represent the scene. Notably, random exploration (*e.g.* sampling a_t from a uniform distribution over the available actions at each time step t) proves to be a strong baseline for this task, performing better than our single-modality RGB agent. Nonetheless, our final agent greatly outperforms the baselines, scoring 0.364 and 0.258 above the random policy and the vanilla curiosity-based agent respectively.

Qualitative Analysis. In Fig. 3.3, we report some top-down views from the testing scenes, together with the trajectory from three different navigation agents: the random baseline, our approach without the penalty for repeated action described in Sec. 3.3.1, and our full model. We notice that the agent without penalty usually remains in the starting area and thus has some difficulties in exploring the whole environment. Instead, our complete model demonstrates better results as it is able to explore a much wider area within the environment. Thus, we conclude that the addition of a penalty for repeated actions in the final reward function is of central importance when it comes to stimulating the agent towards the exploration of regions far from the starting point.

3.5.2 Speaker Results

Here, we provide quantitative and qualitative results for our speaker module, which is composed of a policy and a captioner. The policy is in charge of deciding when to activate the captioner, which in turns generates a description of the first-person view of the agent. Results are reported in Table 3.2 and discussed below.

Speaker Policy. Among the three different policies, the object-driven speaker performs the best in terms of coverage and diversity. In particular, setting a low threshold (**O** ≥ 1) provides the highest scores. At the same time, the agent tends

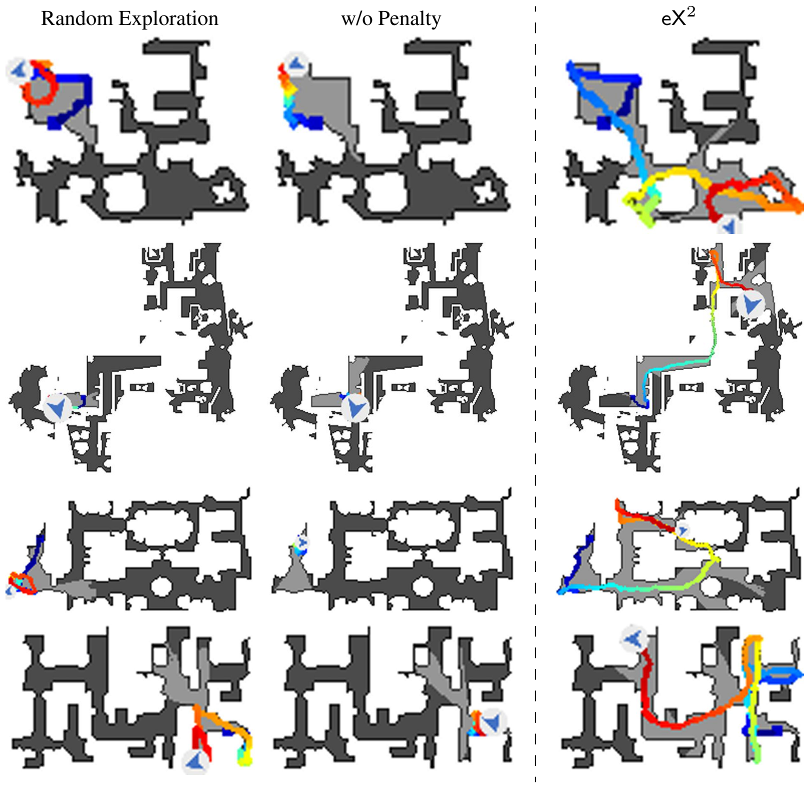


Figure 3.3: Agent trajectories in sample navigation episodes.

to speak more often, which is desirable in a visually rich environment. As the threshold for **O** gets higher, performances get worse. This indicates that, as the number of object in the scene increases, there are many details that the captioner cannot describe. The same applies for the depth-driven policy: while the agent tends to describe well items that are closer, it experiences some troubles when facing an open space with more distant objects ($\mathbf{D} \geq 0.75$).

Instead, our curiosity-driven speaker shows a more peculiar behaviour: as the threshold grows, results get better in terms of diversity, while the coverage scores are quite stable (only -0.005% in terms of Cov$_{>1\%}$). It is also worth mentioning that our curiosity-based speaker can be adopted in any kind of environment, as the driving metric is computed from the raw RGB-D input. The same does not apply in an object-driven policy, since the agent needs semantic information. Further, the curiosity-driven policy employs a learned metric, hence being more related to the exploration module.

	Object-driven policy ($O \geq 1$) Loquacity = 43.3					Object-driven policy ($O \geq 3$) Loquacity = 27.4					Object-driven policy ($O \geq 5$) Loquacity = 15.8				
Captioning Module	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div
eX^2 (6 lay.)	0.456	0.550	0.609	0.706	0.386	0.387	0.502	0.576	0.696	0.363	0.348	0.468	0.549	0.691	0.352
eX^2 (3 lay.)	0.474	0.558	0.612	0.701	0.372	0.384	0.497	0.571	0.691	0.350	0.347	0.467	0.546	0.688	0.338
eX^2 (2 lay.)	**0.485**	**0.579**	**0.637**	**0.727**	0.368	**0.416**	**0.534**	**0.607**	**0.721**	0.349	**0.373**	**0.497**	**0.577**	**0.713**	0.340
eX^2 (1 lay.)	0.468	0.564	0.623	0.720	**0.394**	0.400	0.519	0.593	0.713	**0.377**	0.356	0.479	0.560	0.702	**0.373**
	Depth-driven policy ($D > 0.25$) Loquacity = 38.5					Depth-driven policy ($D > 0.5$) Loquacity = 31.1					Depth-driven policy ($D > 0.75$) Loquacity = 14.8				
Captioning Module	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div
eX^2 (6 lay.)	0.433	0.532	0.600	0.705	0.360	0.420	0.519	0.585	0.701	0.346	0.399	0.497	0.566	0.691	0.339
eX^2 (3 lay.)	0.427	0.524	0.588	0.700	0.349	0.413	0.511	0.577	0.695	0.335	0.394	0.491	0.559	0.685	0.330
eX^2 (2 lay.)	**0.463**	**0.562**	**0.625**	**0.730**	0.341	**0.449**	**0.550**	**0.612**	**0.726**	0.330	**0.425**	**0.525**	**0.595**	**0.715**	0.325
eX^2 (1 lay.)	0.448	0.548	0.613	0.723	**0.371**	0.434	0.536	0.603	0.719	**0.359**	0.412	0.513	0.583	0.708	**0.355**
	Curiosity-driven policy ($S > 0.7$) Loquacity = 27.2					Curiosity-driven policy ($S > 0.85$) Loquacity = 18.2					Curiosity-driven policy ($S > 1.0$) Loquacity = 6.4				
Captioning Module	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div	$\text{Cov}_{>1\%}$	$\text{Cov}_{>3\%}$	$\text{Cov}_{>5\%}$	$\text{Cov}_{>10\%}$	Div
eX^2 (6 lay.)	0.425	0.523	0.588	0.703	0.356	0.421	0.515	0.581	0.699	0.360	0.422	0.518	0.583	0.702	0.364
eX^2 (3 lay.)	0.418	0.514	0.578	0.694	0.348	0.413	0.506	0.571	0.691	0.350	0.413	0.506	0.570	0.690	0.361
eX^2 (2 lay.)	**0.453**	**0.552**	**0.617**	**0.726**	0.340	**0.448**	**0.545**	**0.611**	**0.724**	0.342	**0.448**	**0.545**	**0.610**	**0.723**	0.349
eX^2 (1 lay.)	0.438	0.539	0.604	0.719	**0.370**	0.433	0.530	0.597	0.716	**0.373**	0.434	0.532	0.597	0.717	**0.380**

Table 3.2: Coverage and diversity results for different version of our captioning module. Results are reported for our three speaker policies using different thresholds to determine the agent's loquacity inside the episode.

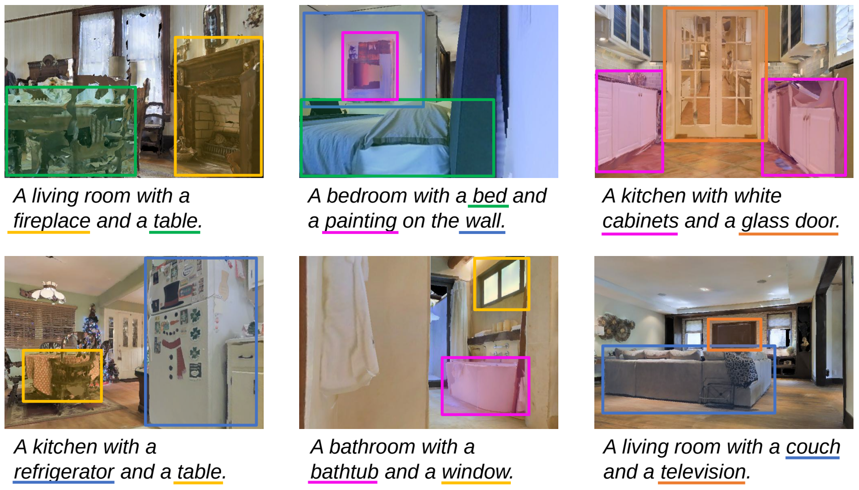

A living room with a fireplace and a table.
A bedroom with a bed and a painting on the wall.
A kitchen with white cabinets and a glass door.
A kitchen with a refrigerator and a table.
A bathroom with a bathtub and a window.
A living room with a couch and a television.

Figure 3.4: Sentences generated on sample images extracted from eX^2 navigation trajectories. For each image, we report the relevant objects present on the scene and we underline their mentions in the caption.

From all these observations, we can conclude that curiosity not only helps training navigation agents, but also represents and important metric when bridging cross-modal components in embodied agents.

Captioner. When evaluating the captioning module, we compare the performance using a different number of encoding and decoding layers. As it can be seen from Table 3.2, the captioning model achieves the best results when composed of 2 layers for coverage and 1 layer for diversity. While this is in contrast with traditional Transformer-based models [164], that employ 6 or more layers, it is in line with recent research on image captioning [42], which finds beneficial to adopt fewer layers. At the same time, a more lightweight network can possibly be embedded in many embodied agents, thus being more appropriate for our task.

Qualitative Analysis. We report some qualitative results for eX^2 in Fig. 3.4. To ease visualization, we underline the items mentioned by the captioner in the sentence, and highlight them with a bounding box of the same color in the corresponding input image. Our agent can explain the scene perceived from a first-person, egocentric point of view. We can notice that eX^2 identifies all the main objects in the environment and produces a suitable description even when the view is partially occluded.

3.6 Conclusion

In this Chapter, we have presented a new setting for Embodied AI that is composed of two tasks: exploration and captioning. Our agent eX^2 uses intrinsic rewards applied to navigation in a photorealistic environment and a novel speaker module that generates captions. The captioner produces sentences according to a speaker policy that could be based on three metrics: object-driven, depth-driven, and curiosity-driven. The experiments show that eX^2 is able to generalize to unseen environments in terms of exploration, while the speaker policy functions to filter the number of time steps where the caption is actually generated. We hope that our work serves as a starting point for future research on this new coupled-task of exploration and captioning. Our results with curiosity-based navigation in photorealistic environments and with the speaker module motivate further works in this direction.

Chapter 4

Spot the Difference: Embodied Agents in Changing Environments

Embodied AI is a recent research area that aims at creating intelligent agents that can move and operate inside an environment. Existing approaches in this field demand the agents to act in completely new and unexplored scenes. However, this setting is far from realistic use cases that instead require executing multiple tasks in the same environment. Even if the environment changes over time, the agent could still count on its global knowledge about the scene while trying to adapt its internal representation to the current state of the environment. To make a step towards this setting, we propose *Spot the Difference*: a novel task for Embodied AI where the agent has access to an outdated map of the environment and needs to recover the correct layout in a fixed time budget. To this end, we collect a new dataset of occupancy maps starting from existing datasets of 3D spaces and generating a number of possible layouts for a single environment. This dataset can be employed in the popular Habitat simulator and is fully compliant with existing methods that employ reconstructed occupancy maps during navigation. Furthermore, we propose an exploration policy that can take advantage of previous knowledge of the environment and identify changes in the scene faster and more effectively than existing agents. Experimental results show that the proposed architecture outperforms existing state-of-the-art models for exploration on this new setting.

This Chapter is related to publication [10], as reported in the List of Publications.

4.1 Introduction

Imagine you have just bought a personal robot, and you ask it to bring you a cup of tea. It will start roaming around the house while looking for the cup. It probably will not come back until some minutes, as it is new to the environment. After the robot knows your house, instead, you expect it to perform navigation tasks much faster, exploiting its previous knowledge of the environment while adapting to possible changes of objects, people, and furniture positioning. Embodied AI has recently gained a lot of attention from the research community, with amazing results in challenging tasks such as visual exploration [136] and navigation [33, 93, 135]. However, in the current setting, the environment is completely unknown to the agent as a new episode begins. We believe that this choice is not supported by real-world experience, where information about the environment can be stored and reused for future tasks. As agents are likely to stay in the same place for long periods, such information may be outdated and inconsistent with the actual layout of the environment. Therefore, the agent also needs to discover those differences during navigation. In this Chapter, we introduce a new task for Embodied AI, which we name *Spot the Difference*. In the proposed setting, the agent must identify all the differences between an outdated map of the environment and its current state – a challenge that combines visual exploration using monocular images and embodied navigation with spatial reasoning. To succeed in this task, the agent needs to develop efficient exploration policies to focus on likely changed areas while exploiting priors about objects of the environment. We believe that this task could be useful to train agents that will need to deal with changing environments.

Recent work on Embodied AI has tackled the training of embodied agents capable of navigating and locating objects [30, 33, 93, 170]. One of the key factors for success in the field consists in building map representations in which knowledge about the environment can be stored while the agent proceeds [32, 135]. However, the dominant training and evaluation protocol involves an agent initialized from scratch that sees the environment for the first time [3]. Another line of work [36, 86, 108, 114, 136], instead, introduces a mapping phase of the environment to increase the performance on both exploration and down-stream tasks. Unfortunately, if the environment changes over time, the agent needs to rebuild a full representation from scratch and cannot count on an efficient policy to update its internal representation of the environment. In this Chapter, we simulate the natural evolution of an environment and design a specific policy to navigate in changing environments seamlessly.

Due to the high cost of 3D acquisitions from the real world, the existing datasets of photorealistic 3D spaces [31, 175] do not contain different layouts for the same environment. In this Chapter, we create a reproducible set-up to generate alternative layouts for an environment. We semi-automatically build a dataset of 2D semantics occupancy maps in which the objects can be removed and rearranged

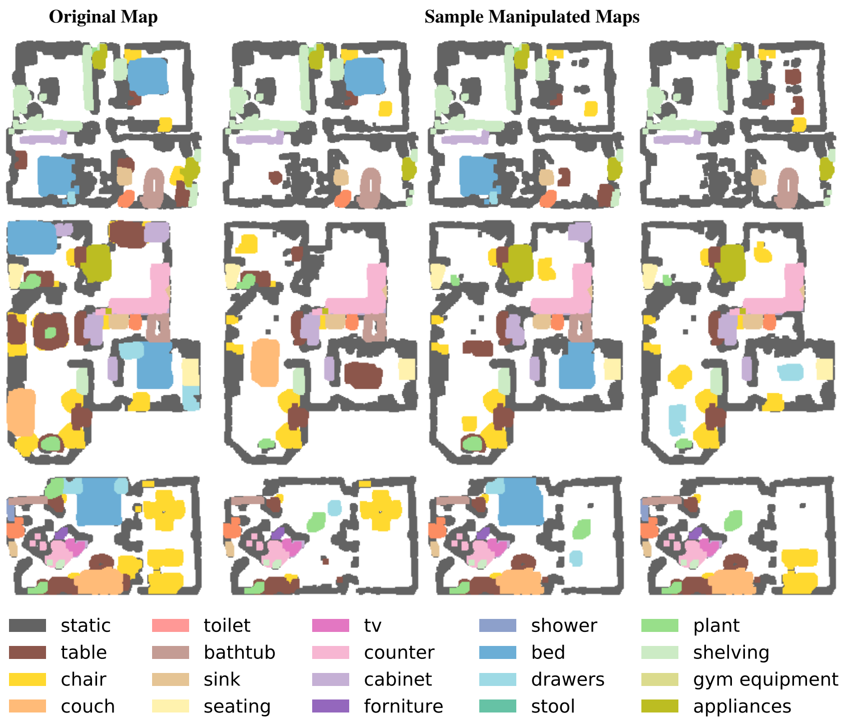


Figure 4.1: Generation of alternative states of an environment: original and sample manipulated semantic maps.

while the area and the position of architectural elements do not change (Fig. 4.1). In the proposed setting, the agent is deployed in an interactive 3D environment and provided with a map from our produced dataset, which represents the information retained while performing tasks in a past state of the environment.

To train agents that can deal with changing environments efficiently, we develop a novel reward function and an approach for navigation aiming at finding relevant differences between the previous layout of the environment and the current one. Our method is based on the recent Active Neural SLAM paradigm proposed in [32] and [135]. Differently from previous proposals, though, it can read and update the given map to identify relevant differences in the environment in the form of their projections on the map. Our dataset and architecture can be employed with the Habitat simulator [145], a popular research platform for Embodied AI that renders photorealistic scenes and that enables seamless sim-to-real deployment of navigation agents [85]. Experimental results show that our approach performs better than existing state-of-the-art architectures for exploration in our newly-proposed task. We also compare with different baselines and evaluate our agent in terms of percentage of area seen, percentage of discovered differences, and metric curves at varying exploration time budgets.

4.2 Related Work

Current research directions on Embodied AI for navigation agents can be categorized according to the quantity of knowledge about the environment provided to the agent prior to performing the task [3]. The first direction focuses on the scenario in which the agent is deployed in a completely new environment for which it has no prior knowledge [32, 70, 86]. Running exploration in parallel with a target-driven navigation task resulted in an effective approach to solve the latter (*e.g.*, object-goal navigation [33] and point-goal navigation [135]). Other directions consider the case in which the agent can exploit pre-acquired information about the environment [34, 43] when performing a navigation task. Such pre-acquired information can be either partial [144, 151, 182] or complete [30, 36, 136]. A major limitation of such approaches is that the obtained map representation is assumed to conform perfectly with the environment where the down-stream task will be performed.

In this Chapter, we explore a fourth direction, in which the pre-acquired map provided to the agent is incomplete or incorrect due to changes occurred in the environment over time. Common strategies to deal with changing environments entail disregarding dynamic objects as landmarks when performing SLAM [25, 143] and applying local policies to avoid them when navigating [111]. An alternative strategy is learning to predict geometric changes based on experience, as done in [119], where the environment is represented as a traversability graph. The main limitation of this strategy is its computational intractability when considering dense metric maps of wide areas, as in our case.

4.3 Proposed Setting

In the first part of this Section, we introduce a new task for embodied agents, named *Spot the Difference*. We then describe the newly-proposed dataset that we create to enable this setting. Finally, we propose an architecture for embodied agents to tackle the defined task.

4.3.1 Spot the Difference: Task Definition

At the beginning of an episode, the agent is spawned in a 3D environment and is given a pre-built occupancy map M, representing its spatial knowledge of the environment, *i.e.* a previous state of the environment that is now obsolete:

$$M = (m_{ij}) \in [0, 1], \quad 0 \leq i, j < W, \tag{4.1}$$

where m_{ij} represents the probability of finding an obstacle at coordinates (i, j). The task entails exploring the current environment to recognize all the differences with respect to the state in which M was computed, in the form of free and occupied space. To accomplish the task, the agent should build a correct occupancy map of the current environment starting from M, recognizing and focusing on parts that are likely to change (*e.g.*, the middle of wide rooms rather than tight corridors).

For every episode of *Spot the Difference*, the agent is given a time budget of T time-steps. At time $t = 0$, the agent holds the starting map representation M. At each time-step t, the map is updated depending on the current observation to obtain M_t. Whenever the agent discovers a new object or a new portion of free space, the internal representation of the map changes accordingly. The goal is to gather as much information as possible about changes in the environment by the end of the episode. To measure the agent performance, we compare the final map M_T produced by the agent with the ground-truth occupancy map M^*. In this sense, the paradigm we adopt is the one of knowledge reuse starting from partial knowledge.

4.3.2 Dataset Creation

Semantic Occupancy Map. Given a 3D environment, we place the agent in a free navigable location with heading $\theta = 0°$ (facing eastward). We assume that the input consists of a depth image and a semantic image and that the camera intrinsics K are known. To build the Semantic Occupancy Map (SOM) of an environment, we project each semantic pixel of the acquired scene into a 2-dimensional top-down map: given a pixel with image coordinates (i, j) and depth value $d_{i,j}$, we first recover its coordinates (x, y, z) with respect to the agent position. Then, we compute the corresponding (u, v) pixel in map through an orthographic projection, using the information about the agent position and heading:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = d_{i,j} K^{-1} \begin{bmatrix} i \\ j \\ 1 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} u \\ v \\ 0 \\ 1 \end{bmatrix} = P_v \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{4.2}$$

We perform the same operation after rotating the agent by $\Delta_\theta = 30°$ until we perform a span from $0°$ to $180°$. To cover the whole scene, we repeat this procedure placing the agent at a distance of $0.5m$ from the previous capture point, following

Dataset Split	Semantic Classes	Scans	Generated SOMs	Episodes
MP3D Train	42	58	2070	≈ 4.5M
MP3D Validation	42	9	160	320
MP3D Test	42	14	260	610
Gibson Validation	20	5	130	450

Table 4.1: Number of manipulated maps generated per dataset split.

the axis directions. The agent elevation is instead kept fixed. During this step, we average the results of subsequent observations of overlapping portions of space.

After the acquisition, we obtain a SOM with C channels, where each pixel corresponds to a $5cm \times 5cm$ portion of space in the 3D environment. For each channel $c \in \{0, ..., C\}$, the map values represent the probability that the corresponding portion of space is occupied by an object of semantic class c.

Multiple Semantic Occupancy Maps for the Same Environment. The SOMs obtained in the previous step can be seen as one possible layout for the corresponding 3D environments. In order to create a dataset with different states (*i.e.* different layouts) of the same environment, instead of manipulating the real-world 3D scenes (changing the furniture position, removing chairs, *etc.*), we propose to modify the SOM to create a set of plausible and different layouts for the environment.

First, we isolate the objects belonging to each semantic category by using an algorithm for connected component labeling [66]. Then, we sample a subset of objects to be deleted from the map and a subset of objects to be re-positioned in a different free location of the map. During sampling, we consider categories that have a high probability of being displaced or removed in the real world and ignore non-movable semantic categories such as *fireplaces*, *columns*, and *stairs*. After this step, we obtain a new SOM representing a possible alternative state for the environment, which could be very different from the one in which the 3D acquisition was taken. Sample manipulated maps can be found in Fig. 4.1.

Dataset Details. To generate alternative SOMs, we start from the Matterport 3D (MP3D) dataset of spaces [31], which comprises 90 different building scans, and is enriched with dense semantic annotations. We consider each floor in the building and compute the SOM for that floor. For each map, we create 10 alternative versions of that same environment. In this step, we discard the floors that have few semantic objects (*e.g.*, empty rooftops) or that are not fully navigable by the agent. As a result, we retain 249 floors belonging to 81 different buildings, thus generating a total of 2490 different semantic occupancy maps for these floors. Finally, we split the dataset into train, validation, and test subsets.

As an additional test bed, we also build a set of out-of-domain maps (13 floors from 5 spaces) taken from the Gibson dataset [175], enriched with semantic annotations from [11], and manipulated as done for the MP3D dataset. For each SOM, multiple episodes are generated by selecting different starting points. More information about our dataset can be found in Table 4.1.

4.3.3 Agent Architecture

Our model for embodied navigation in changing environments comprises three major components: a mapper module, a pose estimator, and a navigation policy (which, in turn, consists of a global policy, a planner, and a local policy). An overview of the proposed architecture is shown in Fig. 4.2 and described below. Although the data we provide is enriched with semantic labels, our agent does not make use of such information directly. This is in line with current state-of-the-art architectures for embodied exploration that we choose as competitors.

Mapper. The mapper module takes as inputs an RGB observation o_t^r and the corresponding depth image o_t^d, representing the first-person view of the agent at time-step t, and outputs the agent-centric occupancy map v_t of a $V \times V$ region in front of the camera. Each pixel in v_t corresponds to a $25mm \times 25mm$ portion of space and consists of two channels containing the probability of that cell being occupied and explored, respectively. As a first step, we encode o_t^r using the first two blocks of ResNet-18 pre-trained on ImageNet, followed by a three-layer CNN. We project the depth image o_t^d using the camera intrinsics [36] and obtain a preliminary map for the visible occupancy. We name the obtained feature representations $\hat{o}_t^r$ and $\hat{o}_t^d$, respectively. We then encode the two feature maps using a U-Net [140]:

$$f_\mu(\hat{o}_t^r, \hat{o}_t^d) = \text{U-Net}_{\text{enc}}(\hat{o}_t^r, \hat{o}_t^d, \mu), \tag{4.3}$$

and decode the $2 \times V \times V$ matrix of probabilities as:

$$v_t = \sigma(\text{U-Net}_{\text{dec}}(f_\mu(\hat{o}_t^r, \hat{o}_t^d), \phi)), \tag{4.4}$$

where μ and ϕ represent the learnable parameters in the U-Net encoder and decoder, respectively, and σ is the sigmoid activation function.

The computed agent-centric occupancy map v_t is then registered in the global occupancy map M_{t-1} coming from the previous time-step to obtain M_t. To that end, we use a geometric transformation to project v_t in the global coordinate system, for which we need a triple (x, y, θ) corresponding to the agent position and heading in the environment. This triple is estimated by a specific component that tracks the agent displacements across the environment, as discussed in the following paragraph.

Pose Estimator. The agent can move across the environment using three actions: *go forward 0.25m*, *turn left 10°*, *turn right 10°*. Since each action may produce a

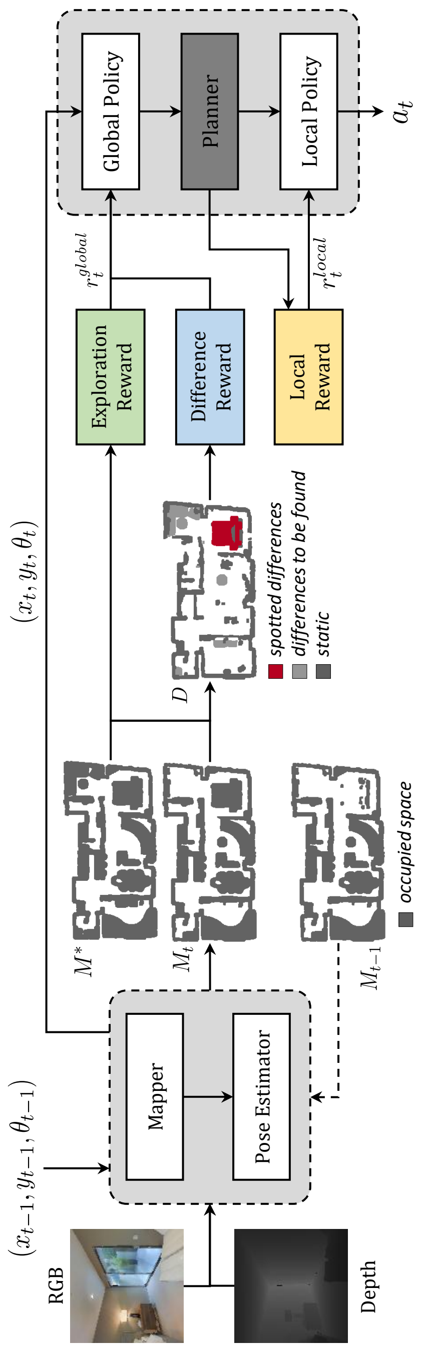


Figure 4.2: Overview of the proposed approach for navigation in changing environments.

different outcome because of physical interactions with the environment (*e.g.*, bumping into a wall) or noise in the actuation system, the pose estimator is used to estimate the real displacement made at every time-step. This module takes as input two consecutive RGB and depth observations, consisting in two pairs (o^r_{t-1}, o^d_{t-1}) and (o^r_t, o^d_t). Additionally, it accepts as input the agent-centric occupancy maps (v_{t-1}, v_t) computed by the mapper at $t-1$ and t. For each modality, we encode information using a CNN followed by a fully-connected layer. We call these intermediate representations $\bar{o}^r_t$, $\bar{o}^d_t$, and $\bar{v}_t$. Then, we compute a first estimate of the relative displacement in terms of (x, y, θ) coordinates and heading for each modality:

$$g(\star) = W_1 \text{max}(W_2 \star + b_2, 0) + b_1, \tag{4.5}$$

with $\star \in \{\bar{o}^r_t, \bar{o}^d_t, \bar{v}_t\}$. We stack the vectors computed in Eq. 4.5 to obtain a 3×3 matrix G. Finally, we compute the agent displacement at time-step t $(\Delta x_t, \Delta y_t, \Delta\theta_t)$ as:

$$(\Delta x_t, \Delta y_t, \Delta\theta_t) = \sum_{i=1}^{3} \alpha_i \cdot G_i, \tag{4.6}$$

$$\alpha_i = \text{softmax}(\text{MLP}_i([\bar{o}^r_t, \bar{o}^d_t, \bar{v}_t])), \tag{4.7}$$

where G_i indicates the i-th row of the G matrix, MLP is a three-layer fully-connected network, and $[\cdot, \cdot, \cdot]$ denotes tensor concatenation. The actual agent position (x_t, y_t, θ_t) is computed iteratively as:

$$(x_t, y_t, \theta_t) = (x_{t-1}, y_{t-1}, \theta_{t-1}) + (\Delta x_t, \Delta y_t, \Delta\theta_t). \tag{4.8}$$

We assume that the agent starting position is the triple $(x_0, y_0, \theta_0) = (0, 0, 0)$.

Global Policy, Planner, and Local Policy. The sampling of atomic actions for the exploration relies on a three-component hierarchical policy. The first component is the global policy, which samples a long-term global goal on the map. An enriched occupancy map $M^+_t \in [0, 1]^{4 \times W \times W}$ is obtained by stacking the occupancy map, the map of visited states, and the one-hot representation of the agent location (x_t, y_t). Then, we compute two versions of M^+_t: one by cropping the map to an agent-centered $G \times G$ area, and the other by max-pooling the map to the same spatial resolution. The 8-channel tensor obtained by concatenating these two versions of M^+_t is fed to the global policy. The global policy consists of a CNN that outputs a probability distribution over the $G \times G$ global action space. We sample the global goal from this distribution, and then transform it in (x, y) global coordinates.

The second component is a planner module, which employs the A* algorithm to decode a local goal on the map. The local goal is an intermediate point, within *0.25m* from the agent, along the trajectory towards the global goal. The last element of our navigation module is the local policy, which decodes the series of atomic actions taking the agent towards the local goal. In particular, the local policy is an RNN decoding the atomic action a_t to execute at every time-step. The reward r_t^{local} given to the local policy is proportional to the reduction in the Euclidean distance d between the agent position and the current local goal:

$$r_t^{local} = d_t - d_{t-1}. \tag{4.9}$$

Following the hierarchical structure, a global goal is sampled every N time-steps. A new local goal is computed if a new global goal is sampled, if the previous local goal is reached, or if the local goal location is known to be not traversable.

Exploiting Past Knowledge for Efficient Navigation. The global policy is trained using a two-term reward. The first term encourages exhaustive exploration and is proportional either to the increase of area-coverage [36] or to the increase of anticipated map accuracy as in [135]. Intuitively, the agent strives to maximize the portion of the seen area and thus maximizes the knowledge gathered during exploration. Moreover, since we consider a setting where a significant amount of knowledge is already available to the agent, we add a reward term to guide the agent towards meaningful points of the map. These correspond to the coordinates where major changes are likely to happen.

Given the occupancy map of the agent at time t, M_t, the true occupancy map for the same environment M^*, and a time budget of T time-steps for exploration, we aim to minimize the following, for $0 < t \leq T$:

$$D = \sum \mathbb{1}[M_t \neq M^*] \tag{4.10}$$

In other words, we want to maximize the number of pixels in the online reconstructed map M_t that the agent correctly shifts from free to occupied (and vice-versa) during exploration. This leads to the reward term for difference discovery:

$$r_{\text{diff}} = \sum \mathbb{1}[M_t = M^*] - \sum \mathbb{1}[M_{t-1} = M^*]. \tag{4.11}$$

The proposed reward term is designed to encourage navigation towards areas in the map that are more likely to contain meaningful differences (*e.g.*, rooms containing more objects that can be displaced or removed from the scene). Additionally, an agent trained with this reward will tend to avoid difficult spots that are likely to produce a mismatch in terms of the predicted occupancy maps. This is because errors in the mapping phase would result in a negative reward.

To train our model, we combine a reward promoting exploration and the more specific reward on found differences to exploit semantic clues in the environment:

$$r_t^{global} = \beta_1 r_{\text{exp}} + \beta_2 r_{\text{diff}} \tag{4.12}$$

where r_{exp} is the reward term encouraging task-agnostic exploration (such as coverage-based or anticipation-based rewards, as described in the next Section), and β_1 and β_2 are two coefficients weighing the importance of the two elements.

4.4 Experiments and Results

In this Section, we detail our experimental setting and show experimental results for our new proposed task.

4.4.1 Experimental Setup

Evaluation Metrics. To evaluate the performance in *Spot the Difference*, we consider three main classes of metrics. First, we consider the percentage of navigable area in the environment seen by the agent during the episode ($\mathsf{Seen}[\%]$). Then, we evaluate the percentage of elements that have been correctly detected as changed in the occupancy map ($\mathsf{Acc.}$) and the pixel-wise Intersection over Union for the *changed* occupancy map elements (IoU). Besides, we evaluate the task as a two-class problem and compute the IoU score for objects that were added in place of free space (IoU_+) and for objects that were deleted during the map creation (IoU_-). In addition, to evaluate the performance independently from the exploration capability, we propose to compute the metrics only on the portion of space that the agent actually visited ($\mathsf{mAcc.}$ and mIoU).

Implementation Details. We conduct our experiment using Habitat [145], a popular platform for Embodied AI in photo-realistic indoor environments [31, 175]. The agent observations are 128×128 RGB-D images from the environment. The learning algorithm adopted for training is PPO [147]. The learning rate is 10^{-3} for the mapper and 2.5×10^{-4} for the other modules. Every model is trained for ≈ 6.5M frames using Adam optimizer [90]. A global goal is sampled every $N = 25$ time-steps. The local and global policies are updated, respectively, every N and $20 \times N$ time-steps, and the mapper is updated every $4 \times N$ time-steps. The size of the local map is $V = 101$, while the global map size is set to $W = 2001$ for the MP3D dataset and to $W = 961$ for the Gibson dataset. The global policy action space size G is 240. The reward coefficients $\{\beta_1, \beta_2\}$ are set to $\{1, 10^{-2}\}$ and $\{1, 10^{-1}\}$ when the exploration reward is based on coverage and anticipation reward, respectively. The length of each episode is fixed to $T = 1000$ time-steps.

Competitors and Baselines. We consider the following competitors and variants of the proposed method on two different setups: one where the agent position is predicted by the agent (as in Eq. 4.8), and one where it has access to oracle coordinates at every time-step.

	MP3D Test																	
	Estimated Localization									**Oracle Localization**								
	Seen[%]	Acc.	IoU$_+$	IoU$_-$	IoU	mAcc.	mIoU$_+$	mIoU$_-$	mIoU	Seen[%]	Acc.	IoU$_+$	IoU$_-$	IoU	mAcc.	mIoU$_+$	mIoU$_-$	mIoU
OccAnt	52.1	26.2	13.4	6.1	11.5	51.1	19.1	8.3	15.8	49.0	35.6	26.5	16.1	24.8	77.8	49.2	23.6	43.2
DR	49.4	29.3	15.3	8.7	13.9	59.7	23.1	11.9	20.2	48.6	37.4	27.2	18.4	26.5	80.1	49.8	27.4	45.8
AR	43.8	30.6	19.7	12.9	18.8	72.5	36.8	18.4	32.7	43.6	32.5	23.2	17.5	23.0	78.7	47.5	26.7	44.5
CR	**53.2**	33.1	18.1	9.6	16.1	65.2	26.4	12.7	22.6	**52.8**	39.2	29.6	18.8	28.0	78.5	51.0	26.6	45.7
AR+DR	51.4	34.5	20.9	12.0	19.3	71.5	33.9	16.2	30.0	51.4	37.8	27.3	18.0	26.2	79.3	48.9	25.8	44.4
CR+DR	52.3	**37.8**	**24.2**	**14.8**	**22.7**	**76.2**	**39.1**	**19.8**	**34.8**	51.8	**40.3**	**29.2**	**19.2**	**28.1**	**82.1**	**50.4**	**26.9**	**46.2**

Table 4.2: Experimental results on MP3D test set. The agent incorporating the proposed reward term for discovered differences outperforms the competitors on the main metrics for the novel Spot the Difference task.

	Gibson Val																	
	Estimated Localization									**Oracle Localization**								
	Seen[%]	Acc.	IoU$_+$	IoU$_-$	IoU	mAcc.	mIoU$_+$	mIoU$_-$	mIoU	Seen[%]	Acc.	IoU$_+$	IoU$_-$	IoU	mAcc.	mIoU$_+$	mIoU$_-$	mIoU
OccAnt	86.2	49.8	11.9	7.2	10.4	58.0	12.3	7.5	10.8	81.6	60.1	**32.1**	21.2	29.2	78.7	**39.6**	22.2	**34.1**
DR	**86.2**	53.2	13.2	8.5	11.7	63.7	13.9	8.8	12.3	**86.1**	**65.2**	30.1	**24.1**	**29.9**	81.1	36.0	25.2	33.8
AR	75.3	51.5	21.4	16.6	20.4	72.7	25.8	17.3	23.3	74.1	53.8	27.9	21.9	27.2	77.0	35.4	23.5	32.7
CR	85.9	57.6	16.7	11.9	15.4	71.3	18.6	12.3	16.7	84.0	62.2	30.6	22.1	28.8	79.5	36.1	23.3	32.8
AR+DR	83.4	58.7	20.0	14.9	19.0	75.8	23.0	15.6	21.1	83.2	63.2	29.6	23.8	29.1	**81.6**	35.8	25.1	33.7
CR+DR	82.1	**60.1**	**24.0**	**19.0**	**23.1**	**78.5**	**27.8**	**19.9**	**25.9**	82.6	63.8	30.3	**24.1**	29.5	**81.6**	36.1	**25.5**	34.0

Table 4.3: Experimental results on Gibson validation set.

Difference Reward (DR): an exploration policy that maximizes the correctly predicted changes between M and M^*. This corresponds to setting $\beta_1 = 0$ and $\beta_2 = 1$ in Eq. 4.12.

Coverage Reward (CR): an agent that explores the environment with an exploration policy that maximizes the covered area and builds the occupancy map as it goes, as reported in [135].

Anticipation Reward (AR): an agent that explores the environment with an exploration policy that maximizes the covered area and the correctly anticipated values in the occupancy map built as it goes, from [135]. Our proposed approach consists of an agent trained with the combination of the difference reward with the coverage reward (*CR+DR*) or with the anticipation reward (*AR+DR*).

Occupancy Anticipation (OccAnt): we also compare with the agent presented by Ramakrishnan *et al.* [135] using the available pre-trained models, referenced to as *OccAnt*. Note that *OccAnt* was trained on the Gibson dataset for the standard exploration task and without any prior map. Thus, it is not directly comparable with the other methods considered. We include it to gain insights into the performance of an off-the-shelf state-of-the-art agent on our task.

4.4.2 Results

Results on MP3D dataset. As a first testbed, we evaluate the different agents on the MP3D *Spot the Difference* test set. We report the results for this experiment in Table 4.2. We observe that the agent combining a reward based on coverage and our reward based on the differences in the environment (*CR+DR*) performs best on all the pixel-based metrics and places second in terms of percentage of seen area. It is worth noting that, even if the results in terms of the area seen are not as high as the ones obtained by the *CR* agent, the addition of our Difference Reward helps the agent to focus on more relevant parts, and thus, it can discover more substantial differences. Additionally, predictions are more accurate and more precise, as indicated by the 4.7% and 6.6% improvements in terms of Acc. and IoU with respect to the *CR* competitor. Instead, a reward based on differences alone is not sufficient to promote good exploration. In fact, although the *DR* agent outperforms the *CR* and *AR* agents on some metrics, our reward alone does not provide as much improvement as when combined with rewards encouraging exploration (as for *CR+DR* and *AR+DR*).

Even in the oracle localization setup, the *CR+DR* agent achieves the best results. Interestingly, the gap with the *CR* agent decreases to 1.1% and 0.1% in terms of Acc. and IoU, respectively. This is because our *CR+DR* agent learns to sample trajectories that can be performed more efficiently and without accumulating a high positioning error. For this reason, the performance boost given by the

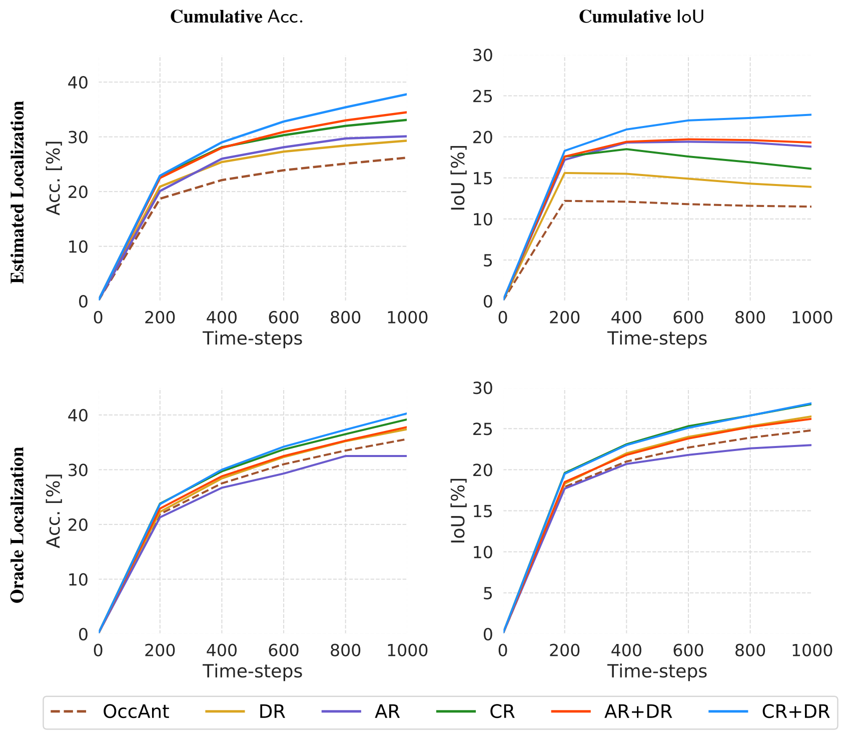


Figure 4.3: Value of accuracy and IoU for the different models at varying time-steps on the MP3D test set.

oracle localization is lower. For both setups, our *CR+DR* agent outperforms the state-of-the-art *OccAnt* agent for exploration on all the metrics.

Finally, in Fig. 4.3, we plot different values of Acc. over different time-steps during the episodes. This way, we can evaluate the whole exploration trend. We can observe that the proposed models incorporating the difference reward outperform the competitors. In particular, the *CR+DR* agent scores first by a significant margin. The performance gap can be noticed even in the first half of the episode and tends to grow with the number of steps.

Results on Gibson dataset. The environments from the Gibson dataset [175] are generally smaller than those in MP3D, and thus, they can be explored more easily and exhaustively. We report the results for this experiment in Table 4.3. Also in this experiment, the *CR+DR* agent performs best on all the metrics but the percentage of the area seen. Although *CR+DR* explores 3.8% of the environment less than the *CR* agent, it still overcomes the competitor by 2.5% and 7.7% in terms of Acc.

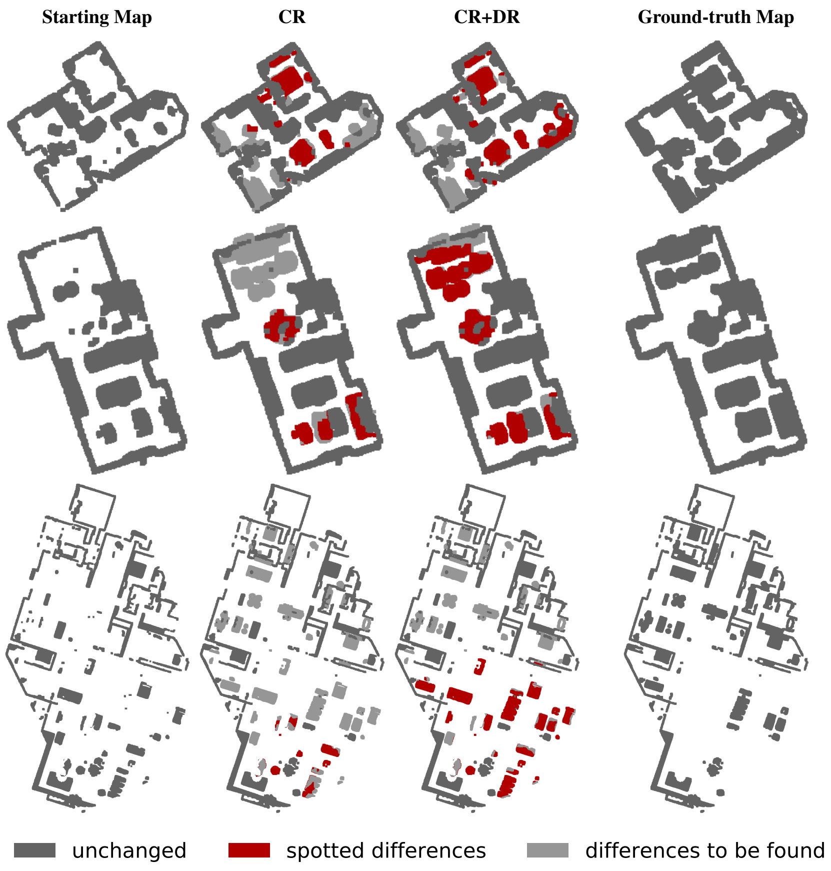


Figure 4.4: Qualitative results comparing the performances of the CR and CR+DR agents for different episodes.

and IoU. The *AR+DR* agent is the second-best in terms of Acc.. The *OccAnt* agent, instead, is competitive in terms of area seen but achieves low Acc. and IoU metrics. As for the oracle localization setup, the agent using only the proposed difference reward (*DR*) performs the best on almost all the metrics. We can conclude that, for small environments, and given an optimal localization system, our reward alone is sufficient to surpass the competitors on *Spot the Difference*.

Qualitative Results. In Fig. 4.4, we report some qualitative results. Starting from the left-most column, we present the starting map given to the agent as the episode begins, the results achieved by the *CR* agent, those of the proposed *CR+DR* agent, and the ground-truth map. The differences that the agents have correctly identified during the episode are highlighted in red. As it can be seen, the *CR+DR* agent can identify more differences than the *CR* counterpart, even in small environments (top row). As the size of the environments grows (bottom row), the performance gap increases and the *CR+DR* agent outperforms its competitor.

4.5 Conclusion

In this Chapter, we proposed *Spot the Difference*: a new task for navigation agents in changing environments. In this novel setting, the agent has to find all variations that occurred in the environment with respect to an outdated occupancy map. Since current datasets of 3D spaces do not account for such variety, we collected a new dataset containing different layouts for the same environment. We tested two state-of-the-art exploration agents on this task and proposed a novel reward term to encourage the discovery of meaningful information during exploration. The proposed agent outperforms the competitors and can identify changes in the environment more efficiently. We believe that the results presented in this Chapter motivate further research on this new proposed setting for Embodied AI.

Chapter 5

A Focus on Impact for Indoor Exploration

Exploration of indoor environments has recently experienced a significant interest, also thanks to the introduction of deep neural agents built in a hierarchical fashion and trained with Deep Reinforcement Learning (DRL) on simulated environments. Current state-of-the-art methods employ a dense extrinsic reward that requires the complete a priori knowledge of the layout of the training environment to learn an effective exploration policy. However, such information is expensive to gather in terms of time and resources. In this Chapter, we propose to train the model with a purely intrinsic reward signal to guide exploration, which is based on the impact of the robot's actions on its internal representation of the environment. So far, impact-based rewards have been employed for simple tasks and in procedurally generated synthetic environments with countable states. Since the number of states observable by the agent in realistic indoor environments is non-countable, we include a neural-based density model and replace the traditional count-based regularization with an estimated pseudo-count of previously visited states. The proposed exploration approach outperforms DRL-based competitors relying on intrinsic rewards and surpasses the agents trained with a dense extrinsic reward computed with the environment layouts. We also show that a robot equipped with the proposed approach seamlessly adapts to PointGoal navigation and real-world deployment.

This Chapter is related to publication [9], as reported in the List of Publications.

5.1 Introduction

Robotic exploration is the task of autonomously navigating an unknown environment with the goal of gathering sufficient information to represent it, often via a spatial map [155]. This ability is key to enable many downstream tasks such as planning [148] and goal-driven navigation [117, 145, 171]. Although a vast portion of existing literature tackles this problem [32, 36, 123, 135], it is not yet completely solved, especially in complex indoor environments. The recent introduction of large datasets of photorealistic indoor environments [31, 175] has eased the development of robust exploration strategies, which can be validated safely and quickly thanks to powerful simulating platforms [48, 145]. Moreover, exploration algorithms developed on simulated environments can be deployed in the real world with little hyperparameter tuning [23, 85, 163], if the simulation is sufficiently realistic.

Most of the recently devised exploration algorithms exploit deep reinforcement learning (DRL) [183], as learning-based exploration and navigation algorithms are more flexible and robust to noise than geometric methods [36, 121, 136]. Despite these advantages, one of the main challenges in training DRL-based exploration algorithms is designing appropriate rewards. In this Chapter, we propose a new reward function that employs the impact of the agent actions on the environment, measured as the difference between two consecutive observations [134], discounted with a pseudo-count [17] for previously-visited states (see Fig 5.1). So far, impact-based rewards [134] have been used only as an additional intrinsic reward in procedurally-generated (*e.g.* grid-like mazes) or singleton (*i.e.* the test environment is the same employed for training) synthetic environments. Instead, our reward can deal with photorealistic non-singleton environments. To the best of our knowledge, this is the first work to apply impact-based rewards to this setting.

Recent research on robot exploration proposes the use of an extrinsic reward based on occupancy anticipation [135]. This reward encourages the agent to navigate towards areas that can be easily mapped without errors. Unfortunately, this approach presents a major drawback, as this reward heavily depends on the mapping phase, rather than focusing on what has been already seen. In fact, moving towards new places that are difficult to map would produce a very low occupancy-based reward. Moreover, the precise layout of the training environments is not always available, especially in real-world applications. To overcome these issues, a different line of work focuses on the design of intrinsic reward functions, that can be computed by the agent by means of their current and past observations. Some examples of recently proposed intrinsic rewards for robot exploration are based on curiosity [22], novelty [136], and coverage [32]. All these rewards, however, tend to vanish with the length of the episode because the agent quickly learns to model the environment dynamics and appearance (for curiosity and novelty-based rewards) or tends to stay in previously-explored areas (for the coverage reward).

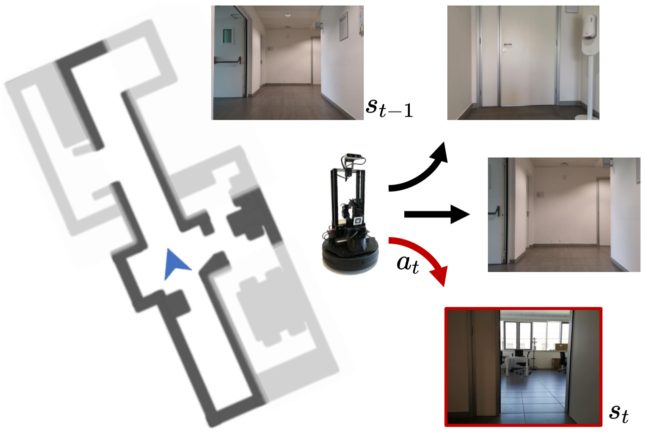


Figure 5.1: The robot is encouraged to take actions that maximize the difference between two consecutive observations.

Impact, instead, provides a stable reward signal throughout all the episode [134].

Since robot exploration takes place in complex and realistic environments that can present an infinite number of states, it is impossible to store a visitation count for every state. Furthermore, the vector of visitation counts would consist of a very sparse vector, and that would cause the agent to give the same impact score to nearly identical states. To overcome this issue, we introduce an additional module in our design to keep track of a pseudo-count for visited states. The pseudo-count is estimated by a density model trained end-to-end and together with the policy. We integrate our newly-proposed reward in a modular embodied exploration and navigation system inspired by that proposed by Chaplot *et al.* [32] and consider two commonly adopted collections of photorealistic simulated indoor environments, namely Gibson [175] and Matterport 3D (MP3D) [31]. Furthermore, we also deploy the devised algorithm in the real world. The results in both simulated and real environments are promising: we outperform state-of-the-art baselines in simulated experiments and demonstrate the effectiveness of our approach in real-world experiments.

5.2 Related Work

Geometric Robot Exploration Methods. Classical heuristic and geometric-based exploration methods rely on two main strategies: frontier-based exploration [178] and next-best-view planning [63]. These methods have been largely used and improved [24, 79, 121] or combined in a hierarchical exploration algorithm [148, 183]. However, when applied with noisy odometry and localization sensors or in highly complex environments, geometric approaches tend to fail [36, 121, 136]. In light of this, increasing research effort has been dedicated to the development of learning-based approaches, which usually exploit DLR to learn robust and efficient exploration policies.

Intrinsic Exploration Rewards. The lack of ground-truth in the exploration task forces the adoption of reinforcement learning (RL) for training exploration methods. Even when applied to tasks different from robot exploration, RL methods have low sample efficiency. Thus, they require designing intrinsic reward functions that encourage visiting novel states or learning the environment dynamics. The use of intrinsic motivation is beneficial when external task-specific rewards are sparse or absent. Among the intrinsic rewards that motivate the exploration of novel states, Bellemare *et al.* [17] introduced the notion of pseudo visitation count by using a Context-Tree Switching (CTS) density model to extract a pseudo-count from raw pixels and applied count-based algorithms. Similarly, Ostrovski *et al.* [124] applied the autoregressive deep generative model PixelCNN [122] to estimate the pseudo-count of the visited state. Recently, Zhang *et al.* [180] proposed a criterion to mitigate common issues in count-based methods. Rewards that encourage the learning of the environment dynamics comprehend Curiosity [129], Random Network Distillation (RND) [29], and Disagreement [130]. Curiosity forces the agent towards areas that maximize to prediction error for future states. RND exploits the prediction error of the state encodings made with a fixed random initialized network. Disagreement employs an ensemble of dynamics models and rewards the agent for visiting states where the disagreement of the ensemble is high. Recently, Raileanu *et al.* [134] proposed to jointly encourage both the visitation of novel states and the learning of the environment dynamics. However, their approach is developed for grid-like environments with a finite number of states, where the visitation count can be easily employed as a discount factor. In this Chapter, we improve Impact, a paradigm that rewards the agent proportionally to the change in the state representation caused by its actions, and design a reward function that can deal with photorealistic scenes with non-countable states.

Learning-based Robot Exploration Methods. In the context of robot exploration and navigation tasks, the introduction of photorealistic simulators has represented a breeding ground for the development of self-supervised DRL-based visual exploration methods. Ramakrishnan *et al.* [136] identified four paradigms for visual exploration: novelty-based, curiosity-based (as defined above), reconstruction-based, and coverage-based. Each paradigm is characterized by a different reward function used as a self-supervision signal for optimizing the exploration policy. In particular, novelty discourages re-visiting the same areas as it is defined as the inverse visitation counts for each area; reconstruction favors reaching positions from which it is easier to predict unseen observations of the environment; and coverage maximizes the information gathered at each time-step, being it the number of objects or landmarks reached or area seen. A coverage-based reward, considering the area seen, is also used in the modular approach to Active SLAM presented in [32], which combines a neural mapper module with a hierarchical navigation policy. To enhance exploration efficiency in complex environments, Ramakrishnan *et al.* [135] resorted to an extrinsic reward by introducing the occupancy anticipation reward, which aims to maximize the agent accuracy in predicting occluded unseen areas.

Deep Generative Models. Deep generative models are trained to approximate high-dimensional probability distributions by means of a large set of training samples. In recent years, literature on deep generative models followed three main approaches: latent variable models like VAE [91], implicit generative models like GANs [64], and exact likelihood models. Exact likelihood models can be classified in non-autoregressive flow-based models, like RealNVP [52] and Flow++ [76], and autoregressive models, like PixelCNN [122] and Image Transformer [127]. Non-autoregressive flow-based models consist of a sequence of invertible transformation functions to compose a complex distribution modeling the training data. Autoregressive models decompose the joint distribution of images as a product of conditional probabilities of the single pixels. Usually, each pixel is computed using as input only the previous predicted ones, following a raster scan order. In this Chapter, we employ PixelCNN [122] to learn a probability distribution over possible states and estimate a pseudo visitation count.

5.3 Proposed Method

5.3.1 Exploration Architecture

Following the current state-of-the-art architectures for navigation for embodied agents [32, 135], the proposed method comprises three main components: a CNN-based mapper, a pose estimator, and a hierarchical navigation policy. The navigation policy defines the actions of the agent, the mapper builds a top-down map of the environment to be used for navigation, and the pose estimator locates the position of the agent on the map. Our architecture is depicted in Fig. 5.2 and described below.

Mapper. The mapper generates a map of the free and occupied regions of the environment discovered during the exploration. At each time step, the RGB observation o_t^{rgb} and the depth observation o_t^{d} are processed to output a two-channel $V \times V$ local map l_t depicting the area in front of the agent, where each cell describes the state of a 5×5 cm area of the environment, the channels measure the probability of a cell being occupied and being explored, as in [32].

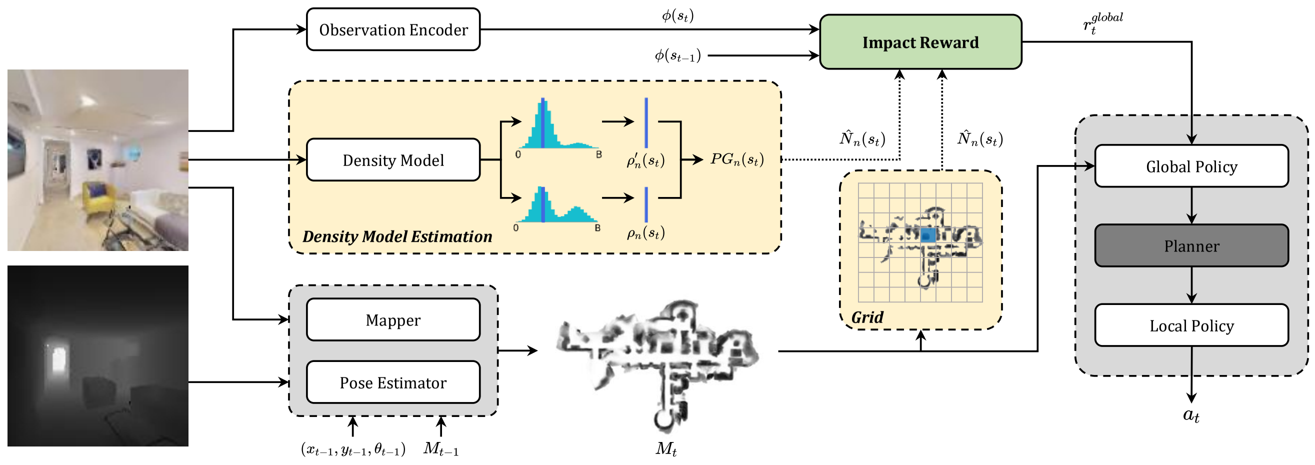


Figure 5.2: Our modular exploration architecture consists of a Mapper that iteratively builds a top-down occupancy map of the environment, a Pose Estimator that predicts the pose of the robot at every step, and a hierarchical self-supervised Navigation Module in charge of sequentially setting exploration goals and predicting actions to navigate towards it. We exploit the impact-based reward to guide the exploration and adapt it for continuous environments, using an Observation Encoder to extract observation features and depending on the method, a Density Model or a Grid to compute the pseudo-count.

Please note that this module performs anticipation-based mapping, as defined in [135], where the predicted local map l_t includes also unseen/occluded portions of space. The local maps are aggregated and registered to the $W \times W \times 2$ global map M_t of the environment using the estimated pose (x_t, y_t, θ_t) from the pose estimator. The resulting global map is used by the navigation policy for action planning.

Pose Estimator. The pose estimator is used to predict the displacement of the agent in consequence of an action. The considered atomic actions a_t of the agent are: *go forward 0.25m, turn left 10°, turn right 10°*. However, the noise in the actuation system and the possible physical interactions between the agent and the environment could produce unexpected outcomes causing positioning errors. The pose estimator reduces the effect of such errors by predicting the real displacement $(\Delta x_t, \Delta y_t, \Delta\theta_t)$. According to [135], the input of this module consists of the RGB-D observations $(o_{t-1}^{rgb}, o_t^{rgb})$ and (o_{t-1}^{d}, o_t^{d}) and the local maps (l_{t-1}, l_t). Each modality $i = 0, 1, 2$ is encoded singularly to obtain three different estimates of the displacement:

$$g_i(e_{t-1}, e_t) = W_1 \text{max}(W_2(e_{t-1}, e_t) + b_2, 0) + b_1, \tag{5.1}$$

where $e_t \in \{o_t^{rgb}, o_t^{d}, l_t\}$ and $W_{1,2}$ and b_2 are weights matrices and bias. Eventually, the displacement estimates are aggregated with a weighted sum:

$$\alpha_i = \text{softmax}(\text{MLP}_i([\bar{o}_t^r, \bar{o}_t^d, \bar{l}_t])), \tag{5.2}$$

$$(\Delta x_t, \Delta y_t, \Delta\theta_t) = \sum_{i=0}^{2} \alpha_i \cdot g_i, \tag{5.3}$$

where MLP is a three-layered fully-connected network, ($\bar{o}_t^r$, $\bar{o}_t^d$, $\bar{l}_t$) are the inputs encoded by a CNN, and $[\cdot, \cdot, \cdot]$ denotes tensor concatenation. The estimated pose of the agent at time t is given by:

$$(x_t, y_t, \theta_t) = (x_{t-1}, y_{t-1}, \theta_{t-1}) + (\Delta x_t, \Delta y_t, \Delta\theta_t). \tag{5.4}$$

Note that, at the beginning of each exploration episode, the agent sets its position to the center of its environment representation, *i.e.*

$$(x_0, y_0, \theta_0) = (0, 0, 0). \tag{5.5}$$

Navigation Module. The sampling of the atomic actions of the agent relies on the hierarchical navigation policy that is composed of the following modules: the global policy, the planner, and the local policy.

The global policy samples a point on an augmented global map of the environment, M_t^+, that represents the current global goal of the agent. The augmented global map M_t^+ is a $W \times W \times 4$ map obtained by stacking the two-channel global map M_t from the Mapper with the one-hot representation of the agent position (x_t, y_t) and the map of the visited positions, which collects the one-hot representations of all the positions assumed by the agent from the beginning of the exploration. Moreover, M_t^+ is in parallel cropped with respect to the position of the agent and max-pooled to a spatial dimension $H \times H$ where $H < W$. These two versions of the augmented global map are concatenated to form the $H \times H \times 8$ input of the global policy that is used to sample a goal in the global action space $H \times H$. The global policy is trained with reinforcement learning with our proposed impact-based reward r_t^{global}, defined below, that encourages exploration.

The planner consists of an A* algorithm. It uses the global map to plan a path towards the global goal and samples a local goal within 1.25m from the position of the agent.

The local policy outputs the atomic actions to reach the local goal and is trained to minimize the euclidean distance to the local goal, which is expressed via the following reward:

$$r_t^{local}(s_t, s_{t+1}) = d(s_{t+1}) - d(s_t), \tag{5.6}$$

where $d(s_t)$ is the euclidean distance to the local goal at time step t. Note that the output actions in our setup are discrete. These platform-agnostic actions can be translated into signals for specific robots actuators, as we do in this Chapter. Alternatively, based on the high-level predicted commands, continuous actions can be predicted, *e.g.* in the form of linear and angular velocity commands to the robot, by using an additional, lower-level policy, as done in [82]. The implementation of such policy is beyond the scope of our work.

Following the hierarchical structure, the global goal is reset every η steps, and the local goal is reset if at least one of the following conditions verifies: a new global goal is sampled, the agent reaches the local goal, the local goal location is discovered to be in a occupied area.

5.3.2 Impact-Driven Exploration

The exploration ability of the agent relies on the design of an appropriate reward for the global policy. In this setting, the lack of external rewards from the environment requires the design of a dense intrinsic reward. To the best of our knowledge, our proposed method presents the first implementation of impact-driven exploration in photorealistic environments. The key idea of this concept is encouraging the agent to perform actions that have impact on the environment and the observations retrieved from it, where the impact at time step t is measured as the l_2-norm of the encodings of two consecutive states $\phi(s_t)$ and $\phi(s_{t+1})$, considering the RGB observation o_t^{rgb} as the state s_t. Following the formulation proposed in [134], the reward of the global policy for the proposed method is calculated as:

$$r_t^{global}(s_t, s_{t+1}) = \frac{\|\phi(s_{t+1}) - \phi(s_t)\|_2}{\sqrt{N(s_{t+1})}}, \tag{5.7}$$

where $N(s_t)$ is the visitation count of the state at time step t, *i.e.* how many times the agent has observed s_t. The visitation count is used to drive the agent out of regions already seen in order to avoid trajectory cycles. Note that the visitation count is episodic, *i.e.* $N_{ep}(s_t) \equiv N(s_t)$. For simplicity, in the following we denote the episodic visitation count as $N(s_t)$.

Visitation Counts. The concept of normalizing the reward using visitation count, as in [134], fails when the environment is continuous, since during exploration is unlikely to visit exactly the same state more than once. In fact, even microscopic changes in terms of translation or orientation of the agent cause shifts in the values of the RGB observation, thus resulting in new states. Therefore, using a photorealistic continuous environment nullifies the scaling property of the denominator of the global reward in Eq. 5.7 because every state s_t during the exploration episode is, most of the times, only encountered for the first time. To overcome this limitation, we implement two types of pseudo-visitation counts $\hat{N}(s_t)$ to be used in place of $N(s_t)$, which extend the properties of visitation counts to continuous environments: *Grid* and *Density Model Estimation*.

Grid: With this approach, we consider a virtual discretized grid of cells with fixed size in the environment. We then assign a visitation count to each cell of the grid. Note that, different from approaches working on procedurally-generated environments like [134], the state space of the environment we consider is continuous also in this formulation, and depends on the pose of the agent (x, y, θ). The grid approach operates a quantization of the agent's positions, and that allows to cluster observation made from similar positions. To this end, we take the global map of the environment and divide it into cells of size $G \times G$. The estimated pose of the agent, regardless of its orientation θ_t, is used to select the cell that the agent occupies at time t. In the *Grid* formulation, the visitation count of the selected cell is used as $N(s_t)$ in Eq. 5.7 and is formalized as:

$$\hat{N}(s_t) = \hat{N}(g(x_t, y_t)), \tag{5.8}$$

where $g(\cdot)$ returns the block corresponding to the estimated position of the agent.

Density Model Estimation (DME): Let ρ be an autoregressive density model defined over the states $s \in S$, where S is the set of all possible states. We call $\rho_n(s)$ the probability assigned by ρ to the state s after being trained on a sequence of states $s_1, ..., s_n$, and $\rho'_n(s)$, or recoding probability [17, 124], the probability assigned by ρ to s after being trained on $s_1, ..., s_n, s$. The prediction gain PG of

ρ describes how much the model has improved in the prediction of s after being trained on s itself, and is defined as

$$PG_n(s) = \log \rho'_n(s) - \log \rho_n(s). \tag{5.9}$$

In this work, we employ a lightweight version of Gated PixelCNN [122] as density model. This model is trained from scratch along with the exploration policy using the states visited during the exploration, which are fed to PixelCNN one at a time, as they are encountered. The weights of PixelCNN are optimized continually over all the environments. As a consequence, the knowledge of the density model is not specific for a particular environment or episode. To compute the input of the PixelCNN model, we transform the RGB observation o^r_t to grayscale and we crop and resize it to a lower size $P \times P$. The transformed observation is quantized to B bins to form the final input to the model, s_t. The model is trained to predict the conditional probabilities of the pixels in the transformed input image, with each pixel depending only on the previous ones following a raster scan order. The output has shape $P \times P \times B$ and each of its elements represents the probability of a pixel belonging to each of the B bins. The joint distribution of the input modeled by PixelCNN is:

$$p(s_t) = \prod_{1}^{P^2} p(\chi_i | \chi_1, ..., \chi_{i-1}), \tag{5.10}$$

where χ_i is the i^{th} pixel of the image s_t. ρ is trained to fit $p(s_t)$ by using the negative log-likelihood loss.

Let $\hat{n}$ be the pseudo-count total, *i.e.* the sum of all the visitation counts of all states during the episode. The probability and the recoding probability of s can be defined as:

$$\rho_n(s) = \frac{\hat{N}_n(s)}{\hat{n}}, \qquad \rho'_n(s) = \frac{\hat{N}_n(s) + 1}{\hat{n} + 1}. \tag{5.11}$$

Note that, if ρ is learning-positive, *i.e.* if $PG_n(s) > 0$ for all possible sequences $s_1, ..., s_n$ and all $s \in S$, we can approximate $\hat{N}_n(s)$ as:

$$\hat{N}_n(s) = \frac{\rho_n(s)(1 - \rho'_n(s))}{\rho'_n(s) - \rho_n(s)} \approx (e^{PG_n(s)} - 1)^{-1}. \tag{5.12}$$

To use this approximation in Eq. 5.7, we still need to address three problems: it does not scale with the length of the episode, the density model could be not learning-positive, and $\hat{N}_n(s)$ should be large enough to avoid the reward becoming too large regardless the goal selection. In this respect, to take into account the length of the episode, we introduce a normalizing factor $n^{-1/2}$, where n is the number of steps done by the agent since the start of the episode. Moreover, to force ρ to be learning-positive, we clip $PG_n(s)$ to 0 when it becomes negative. Finally, to avoid small values at the denominator of r_t^{global} (Eq. 5.7), we introduce a lower bound of 1 to the pseudo visitation count. The resulting definition of $\hat{N}_n(s)$ in the *Density Model Estimation* formulation is:

$$\widetilde{PG}_n = c \cdot n^{-1/2} \cdot (PG_n(s))_+, \tag{5.13}$$

$$\hat{N}_n(s) = \max\left\{\left(e^{\widetilde{PG}_n(s)} - 1\right)^{-1}, 1\right\}, \tag{5.14}$$

where c is a term used to scale the prediction gain. It is worth noting that, unlike the Grid approach that can be applied only when s_t is representable as the robot location, the Density Model Estimation can be adapted to a wider range of tasks, including settings where the agent alters the environment.

5.4 Experimental Setup

Datasets. For comparison with state-of-the-art DRL-based methods for embodied exploration, we employ the photorealistic simulated 3D environments contained in the Gibson dataset [175] and the MP3D dataset [31]. Both these datasets consist of indoor environments where different exploration episodes take place. In each episode, the robot starts exploring from a different point in the environment. Environments used during training do not appear in the validation/test split of these datasets. Gibson contains 106 scans of different indoor locations, for a total of around 5M exploration episodes (14 locations are used in 994 episodes for test in the so-called Gibson Val split). MP3D consists of 90 scans of large indoor environments (11 of those are used in 495 episodes for the validation split and 18 in 1008 episodes for the test split).

Evaluation Protocol. We train our models on the Gibson train split. Then, we perform model selection basing on the results obtained on Gibson Val. We then employ the MP3D validation and test splits to benchmark the generalization abilities of the agents. To evaluate exploration agents, we employ the following metrics. **IoU** between the reconstructed map and the ground-truth map of the environment: here we consider two different classes for every pixel in the map (free or occupied). Similarly, the map accuracy (**Acc**, expressed in m^2) is the portion of the map that has been correctly mapped by the agent. The area seen (**AS**, in m^2) is the total area of the environment observed by the agent. For both the IoU and the area seen, we also present the results relative to the two different classes: free space and occupied space respectively (**FIoU**, **OIoU**, **FAS**, **OAS**). Finally, we report the mean positioning error achieved by the agent at the end of the episode. A larger translation error (**TE**, expressed in m) or angular error (**AE**, in degrees) indicates that the agent struggles to keep a correct estimate of its position throughout the episode. For all the metrics, we consider episodes of length $T = 500$ and $T = 1000$ steps.

For our comparisons, we consider five baselines trained with different rewards. *Curiosity* employs a surprisal-based intrinsic reward as defined in [129]. *Coverage* and *Anticipation* are trained with the corresponding coverage-based and accuracy-based rewards defined in [135]. For completeness, we include two count-based baselines, obtained using the reward defined in Eq. 5.7, but ignoring the contribution of impact (*i.e.* setting the numerator to a constant value of 1). These are *Count (Grid)* and *Count (DME)*. All the baselines share the same overall architecture and training setup of our main models.

Implementation Details. The experiments are performed using the Habitat Simulator [145] with observations of the agent set to be 128×128 RGB-D images and episode length during training set to $T = 500$. Each model is trained with the training split of the Gibson dataset [175] with 40 environments in parallel for ≈ 5M frames.

Navigation Module: The reinforcement learning algorithm used to train the global and local policies is PPO [147] with Adam optimizer and a learning rate of 2.5×10^{-4}. The global goal is reset every $\eta = 25$ time steps and the global action space hyperparameter H is 240. The local policy is updated every η steps and the global policy is updated every 20η steps.

Mapper and Pose Estimator: These models are trained with a learning rate of 10^{-3} with Adam optimizer, the local map size is set with $V = 101$ while the global map size is $W = 961$ for episodes in the Gibson dataset and $W = 2001$ in the MP3D dataset. Both models are updated every 4η time steps, where η is the reset interval of the global policy.

Density Model: The model used for density estimation is a lightweight version of Gated PixelCNN [122] consisting of a 7×7 masked convolution followed by two residual blocks with 1×1 masked convolutions with 16 output channels, a 1×1 masked convolutional layer with 16 output channels, and a final 1×1 masked convolution that returns the output logits with shape $P \times P \times B$, where B is the number of bins used to quantize the model input. We set $P = 42$ for the resolution of the input and the output of the density model, and $c = 0.1$ for the prediction gain scale factor.

5.5 Experimental Results

Exploration Results. As a first step, we perform model selection using the results on the Gibson Val split (Table 5.1). Our agents have different hyperparameters that depend on the implementation for the pseudo-counts. When our model employs grid-based pseudo-counts, it is important to determine the dimension of a single

	Gibson Val (T = 500)								
Model	**IoU ↑**	**FIoU ↑**	**OIoU ↑**	**Acc ↑**	**AS ↑**	**FAS ↑**	**OAS ↑**	**TE ↓**	**AE ↓**
Grid									
G = 2	0.726	0.721	0.730	51.41	61.88	34.17	27.71	0.240	4.450
G = 4	0.796	0.792	0.801	54.34	61.17	33.74	27.42	0.079	1.055
G = 5	**0.806**	**0.801**	**0.813**	**55.21**	**62.17**	**34.31**	**27.87**	**0.077**	**0.881**
G = 10	0.789	0.784	0.794	54.26	61.67	34.06	27.61	0.111	1.434
DME									
B = 64	0.773	0.768	0.778	53.58	61.00	33.79	27.21	0.131	2.501
B = 128	**0.796**	**0.794**	**0.799**	**54.73**	**62.07**	**34.27**	**27.79**	**0.095**	**1.184**
B = 256	0.685	0.676	0.695	49.27	61.40	33.95	27.45	0.311	6.817

Table 5.1: Results for our model selection on Gibson Val for $T = 500$.

cell in this grid-based structure. In our experiments, we test the effects of using $G \times G$ squared cells, with $G \in \{2, 4, 5, 10\}$. The best results are obtained with $G = 5$, with small differences among the various setups. When using pseudo-counts based on a density model, the most relevant hyperparameters depend on the particular model employed as density estimator. In our case, we need to determine the number of bins B for PixelCNN, with $B \in \{64, 128, 256\}$. We find out that the best results are achieved with $B = 128$.

In Table 5.2, we compare the Impact (Grid) and Impact (DME) agents with the baseline agents previously described on the considered datasets. For each model and each split, we test 5 different random seeds and report the mean result for each metric. For the sake of readability, we do not report the standard deviations for the different runs, which we quantify in around 1.2% of the mean value reported.

As it can be seen, results achieved by the two proposed impact-based agents are constantly better than those obtained by the competitors, both for $T = 500$ and $T = 1000$. It is worth noting that our intrinsic impact-based reward outperforms strong extrinsic rewards that exploit information computed using the ground-truth layout of the environment. Moreover, the different implementations chosen for the pseudo-counts affect final performance, with Impact (DME) bringing the best results in terms of AS and Impact (Grid) in terms of IoU metrics. From the results it also emerges that, although the proposed implementations for the pseudo-count in Eq. 5.7 lead to comparable results in small environments as those contained in Gibson and MP3D Val, the advantage of using DME is more evident in large, complex environments as those in MP3D Test.

In Fig. 5.3, we report some qualitative results displaying the trajectories and the area seen by different agents in the same episode. Also from a qualitative point of view, the benefit given by the proposed reward in terms of exploration trajectories and explored areas is easy to identify.

	Gibson Val (T = 500)									Gibson Val (T = 1000)								
Model	**IoU $\uparrow$**	**FIoU $\uparrow$**	**OIoU $\uparrow$**	**Acc $\uparrow$**	**AS $\uparrow$**	**FAS $\uparrow$**	**OAS $\uparrow$**	**TE $\downarrow$**	**AE $\downarrow$**	**IoU $\uparrow$**	**FIoU $\uparrow$**	**OIoU $\uparrow$**	**Acc $\uparrow$**	**AS $\uparrow$**	**FAS $\uparrow$**	**OAS $\uparrow$**	**TE $\downarrow$**	**AE $\downarrow$**
Curiosity	0.678	0.669	0.688	49.35	61.67	34.16	27.51	0.330	7.430	0.560	0.539	0.581	45.71	67.64	37.19	30.45	0.682	14.862
Coverage	0.721	0.715	0.726	51.47	61.13	34.07	27.06	0.272	5.508	0.653	0.641	0.664	50.10	66.15	36.77	29.38	0.492	10.796
Anticipation	0.783	0.778	0.789	54.68	60.96	34.15	26.81	0.100	1.112	0.773	0.763	0.782	56.37	66.61	37.17	29.44	0.155	1.876
Count (Grid)	0.714	0.706	0.721	50.85	61.61	34.17	27.44	0.258	5.476	0.608	0.592	0.624	48.22	67.80	37.31	30.50	0.520	10.996
Count (DME)	0.764	0.757	0.772	52.81	60.69	33.68	27.01	0.148	2.888	0.708	0.694	0.722	52.67	66.91	36.81	30.12	0.282	5.802
Impact (Grid)	**0.803**	**0.797**	**0.809**	54.94	61.90	34.07	27.83	**0.079**	**0.878**	**0.802**	**0.793**	**0.811**	**57.21**	67.74	37.04	30.69	**0.119**	**1.358**
Impact (DME)	0.800	0.796	0.803	**55.10**	**62.59**	**34.45**	**28.14**	0.095	1.166	0.789	0.783	0.796	56.77	**68.34**	**37.42**	**30.92**	0.154	1.958

	MP3D Val (T = 500)									MP3D Val (T = 1000)								
Model	**IoU $\uparrow$**	**FIoU $\uparrow$**	**OIoU $\uparrow$**	**Acc $\uparrow$**	**AS $\uparrow$**	**FAS $\uparrow$**	**OAS $\uparrow$**	**TE $\downarrow$**	**AE $\downarrow$**	**IoU $\uparrow$**	**FIoU $\uparrow$**	**OIoU $\uparrow$**	**Acc $\uparrow$**	**AS $\uparrow$**	**FAS $\uparrow$**	**OAS $\uparrow$**	**TE $\downarrow$**	**AE $\downarrow$**
Curiosity	0.339	0.473	0.205	97.82	118.13	75.73	42.40	0.566	7.290	0.336	0.449	0.223	109.79	157.27	100.07	57.20	1.322	14.540
Coverage	0.352	0.494	0.210	102.05	120.00	76.78	43.21	0.504	5.822	0.362	0.492	0.232	116.58	158.83	100.76	58.07	1.072	11.624
Anticipation	0.381	0.530	0.231	106.02	114.06	72.94	41.13	0.151	1.280	0.420	0.568	0.272	126.86	147.33	93.56	53.78	0.267	2.436
Count (Grid)	0.347	0.488	0.206	99.00	116.77	75.00	41.76	0.466	5.828	0.350	0.474	0.226	112.75	157.13	100.03	57.10	1.074	11.686
Count (DME)	0.359	0.493	0.225	101.73	112.65	72.22	40.43	0.268	3.318	0.379	0.505	0.254	119.07	149.62	95.16	54.46	0.590	6.544
Impact (Grid)	0.383	0.531	**0.234**	107.41	116.60	74.44	42.17	**0.120**	**0.860**	**0.440**	**0.595**	**0.285**	**133.97**	157.19	99.61	57.58	**0.202**	**1.294**
Impact (DME)	**0.396**	**0.560**	0.233	**111.61**	**124.06**	**79.47**	**44.59**	0.232	1.988	0.427	0.587	0.268	133.27	**166.20**	**105.69**	**60.50**	0.461	3.654

	MP3D Test (T = 500)									MP3D Test (T = 1000)								
Model	**IoU $\uparrow$**	**FIoU $\uparrow$**	**OIoU $\uparrow$**	**Acc $\uparrow$**	**AS $\uparrow$**	**FAS $\uparrow$**	**OAS $\uparrow$**	**TE $\downarrow$**	**AE $\downarrow$**	**IoU $\uparrow$**	**FIoU $\uparrow$**	**OIoU $\uparrow$**	**Acc $\uparrow$**	**AS $\uparrow$**	**FAS $\uparrow$**	**OAS $\uparrow$**	**TE $\downarrow$**	**AE $\downarrow$**
Curiosity	0.362	0.372	0.352	109.66	130.48	85.98	44.50	0.620	7.482	0.361	0.365	0.357	130.10	185.36	121.65	63.71	1.520	14.992
Coverage	0.390	0.401	0.379	116.71	134.89	88.15	46.75	0.564	5.938	0.409	0.418	0.399	142.86	193.20	126.21	66.99	1.240	11.814
Anticipation	0.424	0.433	**0.415**	117.87	124.24	81.31	42.93	0.151	1.306	0.484	0.491	0.478	153.83	174.76	114.29	60.47	0.289	2.356
Count (Grid)	0.364	0.381	0.348	117.50	134.85	89.81	45.05	0.525	5.790	0.377	0.391	0.363	144.26	194.76	129.22	65.53	1.246	11.608
Count (DME)	0.391	0.397	0.385	114.02	123.86	81.86	42.00	0.287	3.322	0.418	0.419	0.418	140.21	172.44	113.25	59.19	0.657	6.572
Impact (Grid)	0.420	0.430	0.409	124.44	130.98	86.08	44.90	**0.124**	**0.834**	**0.502**	**0.510**	**0.494**	168.55	190.03	124.44	65.60	**0.218**	**1.270**
Impact (DME)	**0.426**	**0.444**	0.409	**133.51**	**144.64**	**95.70**	**48.94**	0.288	2.312	0.481	0.498	0.464	**174.18**	**212.00**	**140.10**	**71.90**	0.637	4.390

Table 5.2: Exploration results on Gibson Val, MP3D Val, and MP3D Test, at different episode maximum length.

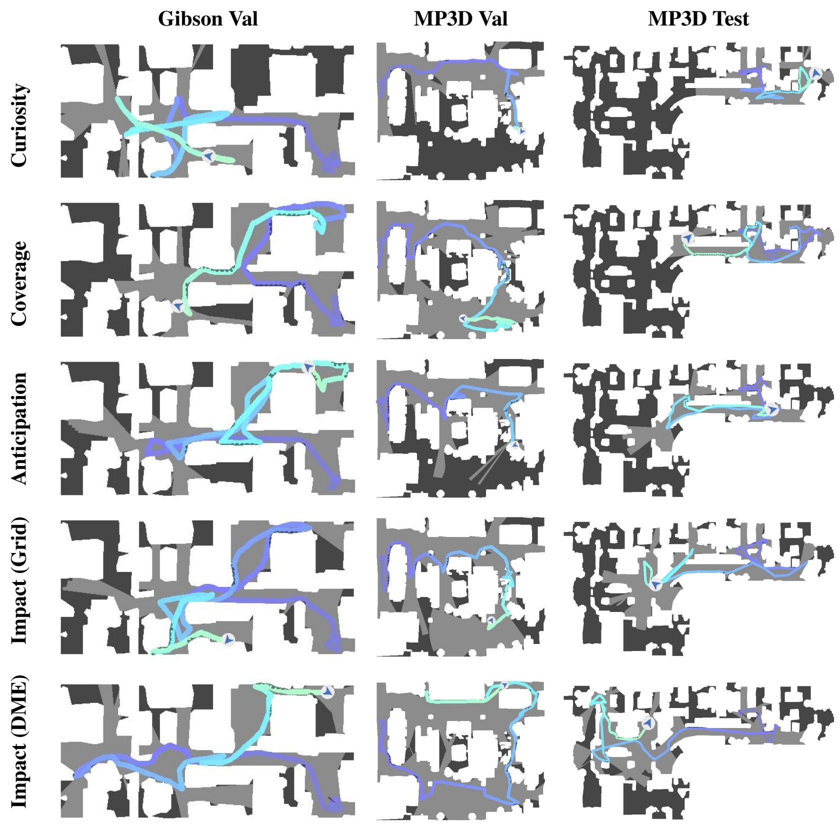


Figure 5.3: Qualitative results. For each model, we report three exploration episodes on Gibson and MP3D validation sets for $T = 500$.

PointGoal Navigation. One of the main advantages of training deep modular agents for embodied exploration is that they easily adapt to perform downstream tasks, such as PointGoal navigation [145]. Recent literature [32, 135] has discovered that hierarchical agents trained for exploration are competitive with state-of-the-art architecture tailored for PointGoal navigation and trained with strong supervision for 2.5 billion frames [171]. Additionally, the training time and data required to learn the policy is much more limited (2 to 3 orders of magnitude smaller). In Table 5.3, we report the results obtained using two different settings. The *noise-free pose sensor* setting is the standard benchmark for PointGoal navigation in Habitat [145]. In the *noisy pose sensor* setting, instead, the pose sensor readings are noisy, and thus the agent position must be estimated as the

	Noise-free Pose Sensor				Noisy Pose Sensor			
Model	**D2G ↓**	**SR ↑**	**SPL ↑**	**sSPL ↑**	**D2G ↓**	**SR ↑**	**SPL ↑**	**sSPL ↑**
OccAnt [135]	-	0.930	0.800	-	-	-	-	-
ANS [32]	-	0.950	0.846	-	-	-	-	-
Curiosity	**0.238**	**0.970**	**0.914**	**0.899**	0.302	0.861	0.822	0.890
Coverage	0.240	**0.970**	0.909	0.895	0.288	0.827	0.788	0.886
Anticipation	0.285	0.965	0.906	0.892	0.309	0.885	0.835	0.884
Impact (Grid)	0.252	0.969	0.908	0.894	**0.226**	**0.923**	**0.867**	**0.893**
Impact (DME)	0.264	0.967	0.907	0.895	0.276	0.913	0.859	**0.893**
DD-PPO [171]	-	*0.967*	*0.922*	-	-	-	-	-

Table 5.3: PointGoal Navigation results on the Validation subset of the Gibson dataset. Underlined denotes second best.

episode progresses. We consider four main metrics: the average distance to the goal achieved by the agent (**D2G**) and three success-related metrics. The success rate (**SR**) is the fraction of episodes terminated within 0.2 meters from the goal, while the **SPL** and SoftSPL (**sSPL**) weigh the distance from the goal with the length of the path taken by the agent in order to penalize inefficient navigation. As it can be seen, the two proposed agents outperform the main competitors from the literature: OccAnt [135] and Active Neural Slam (ANS) [32]. For both the competitors, we report the results achieved by the RGB-D agents, as reported in the papers.

When comparing with our baselines in the noise-free setting, the overall architecture design allows for high-performance results, as the reward influences map estimation only marginally. In fact, in this setting, the global policy and the pose estimation module are not used, as the global goal coincides with the episode goal coordinates, and the agent receives oracle position information. Thus, good results mainly depend on the effectiveness of the mapping module. Instead, in the noisy setting, the effectiveness of the reward used during training influences navigation performance more significantly. In this case, better numerical results originate from a better ability to estimate the precise pose of the agent during the episode. For completeness, we also compare with the results achieved by DD-PPO [171], a method trained with reinforcement learning for the PointGoal task on 2.5 billion frames, 500 times more than the frames used to train our agents.

Real-world Deployment. As agents trained in realistic indoor environments using the Habitat simulator are adaptable to real-world deployment [23, 85], we also deploy the proposed approach on a LoCoBot robot [106]. We employ the PyRobot interface [118] to deploy code and trained models on the robot. To enable the adaptation to the real-world environment, there are some aspects that must be taken into account during training. As a first step, we adjust the simulation in order to reproduce realistic actuation and sensor noise. To that end, we adopt the noise model proposed in [32] based on Gaussian Mixture Models fitting real-world noise data acquired from a LoCoBot. Additionally, we modify the parameters of the RGB-D sensor used in simulation to match those of the RealSense camera mounted on the robot. Specifically, we change the camera resolution and field of view, the range of depth information, and the camera height. Finally, it is imperative to prevent the agent from learning simulation-specific shortcuts and tricks. For instance, the agent may learn to slide along the walls due to imperfect dynamics in simulation [85]. To prevent the learning of such dynamics, we employ the *bump* sensor provided by Habitat and block the agent whenever it is in contact with an obstacle. When deployed in the real world, our agent is able to explore the environment without getting stuck or bumping into obstacles. Further details on sim-to-real adaptation and real-world deployment, as well as experimental results, can be found in Chapter 8.

5.6 Conclusion

In this Chapter, we presented an impact-driven approach for robotic exploration in indoor environments. Different from previous research that considered a setting with procedurally-generated environments with a finite number of possible states, we tackle a problem where the number of possible states is non-numerable. To deal with this scenario, we exploit a deep neural density model to compute a running pseudo-count of past states and use it to regularize the impact-based reward signal. The resulting intrinsic reward allows to efficiently train an agent for exploration even in absence of an extrinsic reward. Furthermore, extrinsic rewards and our proposed reward can be jointly used to improve training efficiency in reinforcement learning. The proposed agent stands out from the recent literature on embodied exploration in photorealistic environments. Additionally, we showed that the trained models can be deployed in the real world.

Chapter 6

Vision-and-Language Navigation with Dynamic Convolution

In Vision-and-Language Navigation (VLN), an embodied agent needs to reach a target destination with the only guidance of a natural language instruction. To explore the environment and progress towards the target location, the agent must perform a series of low-level actions, such as rotate, before stepping ahead. In this Chapter, we propose to exploit dynamic convolutional filters to encode the visual information and the lingual description in an efficient way. Differently from some previous works that abstract from the agent perspective and use high-level navigation spaces, we design a policy which decodes the information provided by dynamic convolution into a series of low-level, agent friendly actions. Results show that our model exploiting dynamic filters performs better than other architectures with traditional convolution, being the new state-of-the-art for embodied VLN in the low-level action space. Additionally, we attempt to categorize recent work on VLN depending on their architectural choices and distinguish two main groups: we call them *low-level actions* and *high-level actions* models. To the best of our knowledge, we are the first to propose this analysis and categorization for VLN.

This Chapter is related to publication [2], as reported in the List of Publications.

6.1 Introduction

Imagine finding yourself in a large conference hall, with an assistant giving you instructions on how to reach the room for your talk. You are likely to hear something like: *turn right at the end of the corridor, head upstairs and reach the third floor: your room is immediately on the left.* Succeeding in the task of finding your target location is rather nontrivial because of the length of the instruction and its sequential nature: the flow of actions must be coordinated with a series of visual examinations – like recognizing the end of the corridor or the floor number. Furthermore, navigation complexity dramatically increases if the environment is unknown, and no prior knowledge, such as a map, is available.

Vision-and-Language Navigation (VLN) [6] is a challenging task that demands an embodied agent to reach a target location by navigating unseen environments, with a natural language instruction as its only clue. Similarly to the previous example, the agent must assess different sub-tasks to succeed. First, a fine-grained comprehension of the given instruction is needed. Then, the agent must be able to map parts of the description into the visual perception. For example, *walking past the piano* requires to find and focus on the piano, rather than considering other objects in the scene. Finally, the agent needs to understand when the navigation has been completed and send a stop signal.

VLN has been first proposed by Anderson *et al.* [6], with the aim of connecting the research efforts on vision-and-language understanding [5, 9, 46, 47, 65, 166, 177] with the raising area of embodied AI [3, 44, 45, 175]. This is particularly challenging, as embodied agents must deal with a series of issues that do not belong to traditional vision and language tasks [3], like contextual decision-making and planning. Recent works on VLN [56, 109, 110, 159] integrate the agent with a simplified action space in which it "*only needs to make high-level decisions as to which navigable direction to go next*" [56]. In this scenario, the agent does not need to infer the sequence of actions to progress in the environment (*e.g.*, turn right 30 degrees, then move forward) but it exploits a navigation graph to teleport itself to an adjacent location. The adoption of this high-level action space allowed for a significant boost in success rates, while partly depriving the task of its embodied nature, and leaving space for little more than pure visual and language understanding. We claim that this type of approach is inconvenient, as it strongly relies on prior knowledge on the environment. Depending on information such as the position and the availability of navigable directions, it reduces the task to a pure graph navigation. Moreover, it ignores the active role of the agent, as it only perceives the surrounding scene and selects the next target viewpoint from a limited set. We claim instead that the agent should be the principal component of embodied VLN [3]. Consequently, the output space should match with the low-level set of movements that the agent can perform.

In this Chapter, we propose a novel architecture for embodied VLN which

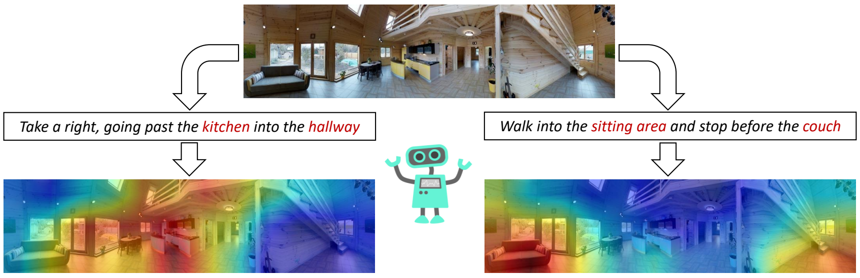

Take a right, going past the kitchen into the hallway
Walk into the sitting area and stop before the couch

Figure 6.1: Given a fixed visual observation, dynamic convolutional filters can extract a subset of specific features depending on the leading instruction. In this example, the agent focuses on two different parts of the same environment (best viewed in color).

employs dynamic convolutional filters [101] to identify the next target direction, without getting any information about the navigable viewpoints from the simulator. Convolutional filters are produced via an attention mechanism which follows the given instruction, and are in turn used to attend relevant directions of the scene towards which the agent should move. We then rely on a policy network to predict the sequence of low-level actions.

Dynamic convolutional filters, proposed by Li *et al.* [101], were first conceived to identify and track people by a natural language specification. They were then successfully employed in other computer vision tasks, such as actor and action video segmentation from a sentence [58]. Nonetheless, these works considered mainly short descriptions, while we deal with complex sentences and long-term dependencies. We generate dynamic filters according to the given instruction, to extract only the relevant information from the visual context. In this way, the same observation can lead to different feature maps, depending on the part of the instruction that the agents must complete (Fig. 6.1).

The proposed method is competitive with prior work that performs high-level navigation exploiting information about the reachable viewpoints (*i.e.* the navigation graph). Additionally, our approach is fully compliant with recent recommendations for embodied navigation [3]. When compared with models that are compliant with the VLN setup, we overcome the current state of the art by a significant margin. To sum up, our contributions are as follows:

- We propose a new encoder-decoder architecture for embodied VLN, which for the first time employs dynamic convolutional filters to attend relevant regions of the visual scene and control the actions of the agent.

- We show, through extensive experimental evaluations, that in a mutable environment with shifting goals dynamic convolutional filters can provide better performance than traditional convolutional filters. Results show that our proposed architecture overcomes the state of the art on the embodied VLN task.

- As a complementary contribution, we categorize previous work on VLN basing on their level of abstraction and generalizability. We distinguish a group of works that strongly relies on the simulating platform and on the navigation graph, we call them *high-level actions* models. A second group, named *low-level actions* models, includes methods that are more agnostic on the underlying implementation and that directly predicts agent actions. With this categorization, we hope to encourage further research to consider low-level and high-level action spaces as distinct fields of application when dealing with VLN.

6.2 Related Work

There is a wide area of research devoted to bridge natural language processing and image understanding. Image captioning [5, 166, 177], visual question answering [9, 65], and visual dialog [46, 47] are examples of active research areas in this field. At the same time, visual navigation [71, 149, 175] and goal-oriented instruction following [35, 57, 133] represent an important part of current work on embodied AI [44, 45, 145, 179]. In this context, Vision-and-Language Navigation (VLN) [6] constitutes a peculiar challenge, as it enriches traditional navigation with a set of visually rich environments and detailed instructions. Additionally, all the scenes are photo-realistic and unknown to the agent beforehand.

Low-level Vision-and-Language Navigation. In low-level VLN, the agent takes move in the environment by using actions such as *rotate*, *tilt up*, and *step ahead*. So far, only a small portion of literature has taken this direction. Anderson *et al.* [6] build on a traditional sequence-to-sequence architecture, while Wang *et al.* [169] employ a mixture of model-free and model-based reinforcement learning. In these works the agent perceives only the first person view of the surrounding environment. In this Chapter we propose a sequence-to-sequence model which exploits dynamic convolution to make the visual representation more compact and informative for the agent. In the proposed architecture, the agent perceives the 360° image of the surroundings. However, this generalization does not hurt adaptability to real-world scenarios.

High-level Vision-and-Language Navigation. The idea of a high-level action space was first proposed by Fried *et al.* [56], and immediately allowed for an important boost in terms of performance. Following work includes visual and textual co-grounding with progress inference [109] and backtracking with learned heuristics [110]. Other methods implement a speaker module which strengthens consistency between the chosen path and the instruction [56, 168]. Wang *et al.* [168] propose a reinforced cross-modal matching critic, together with a new self-supervised imitation learning setting. Tan *et al.* [159] devise a novel environmental dropout method to improve traditional features dropout for VLN. Ke *et al.* [88] propose a FAST navigation agent which improves the performance both over greedy decoding of the next action and over beam search. Very recently, Zhu *et al.* [184] exploit auxiliary reasoning tasks and the rich semantic given by the navigation in their model, while Hao *et al.* [72] investigate an efficient pre-training for generic VLN agents. While pragmatic approaches with high-level reasoning allow for a boost in performance, architectures built for high-level VLN rely heavily on the information coming from the underlying simulating platform. Even when the environment is supposed to be unknown (*e.g.* during test) the agent can get a priori knowledge from the connectivity graph and exploit this information for a more efficient navigation.

6.3 Proposed Method

We propose an encoder-decoder architecture for Vision-and-Language Navigation. Our work employs dynamic convolutional filters conditioned on the current instruction to extract the relevant piece of information from the visual perception, which is in turn used to feed a policy network which controls the actions performed by the agent. The output of our model is a probability distribution over a low-level action space $\boldsymbol{A} = \{a_i\}_{i=1}^{6}$, which comprises the following actions: *turn 30° left*, *turn 30° right*, *raise elevation*, *lower elevation*, *go ahead*, *end episode*. The output probability distribution at a given step, $p_t = P(a_t|X, V_t, h_{t-1})$, depends on a natural language instruction X, the current visual observation V_t, and on the policy hidden state at time step $t-1$. Our architecture is depicted in Fig. 6.2.

6.3.1 Encoder

To represent the two inputs of the architecture, *i.e.*the instruction and the visual input at time t, we devise an instruction and a visual encoder. The instruction encoder provides a representation of the navigation instructions that is employed to guide the whole navigation episode. On the other hand, the visual encoding module operates at every single step, building a representation of the current observation which depends on the agent position.

Instruction Encoding. The given natural language instruction is split into single words via tokenization, and stop words are filtered out to obtain a shorter description. Differently from previous works that train word embeddings from scratch, we rely on word embeddings obtained from a large corpus of documents. Beside

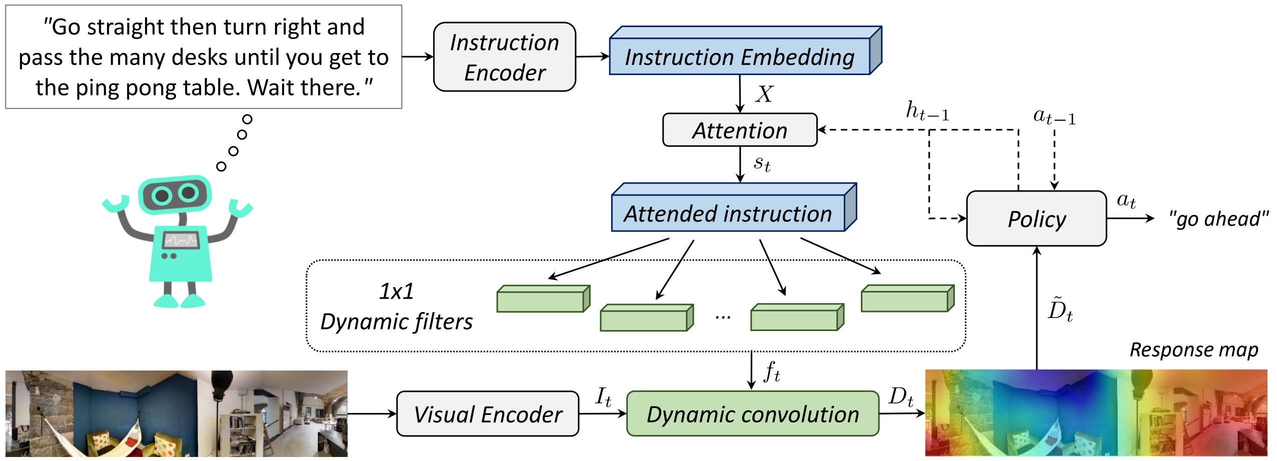


Figure 6.2: Schema of the proposed architecture for VLN. The input instruction is encoded via an attentive mechanism. We then generate dynamic convolutional filters that let the agent attend relevant regions of the input scene. The resulting visual information is used to feed a policy network controlling the movements of the embodied agent in a low-level action space.

providing semantic information which could not be learned purely from VLN instructions, the use of learned word embedding also let us handle words that are not present in the training set (see Sec. 6.4.2 for a discussion). Given an instruction with length N, we denote its embeddings sequence as $L = (l_1, l_2, ..., l_N)$, where l_i indicates the embedding for the i-th word. Then, we adopt a Long Short-Term Memory (LSTM) network to provide a timewise contextual representation of the instruction:

$$X = (x_1, x_2, ..., x_N) = \text{LSTM}(L), \tag{6.1}$$

where each x_i denotes the hidden state of the LSTM at time i, thus leading to a final representation with shape (N, d), where d is the size of the LSTM hidden state.

Visual Features Encoding. As visual input, we employ the panoramic 360° view of the agent, and discretize the resulting equirectangular image in a 12×3 grid, consisting of three different elevation levels and 30° heading shift from each other. Each location of the grid is then encoded via the 2048-dimensional features extracted from a ResNet-152 [73] pre-trained on ImageNet [49]. We also append to each cell vector a set of coordinates relative to the current agent heading and elevation:

$$coord_t = (\sin\phi_t, \cos\phi_t, \sin\theta_t)\,, \tag{6.2}$$

where $\phi_t \in (-\pi, \pi]$ and $\theta_t \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ are the heading and elevation angles *w.r.t.* the agent position. By adding $coord_t$ to the image feature map, we encode information related to concepts such as *right*, *left*, *above*, *below* into the agent observation.

6.3.2 Decoder

Given the instruction embedding X for the whole episode, we use an attention mechanism to select the next part of the sentence that the agent has to fulfill. We denote this encoded piece of instruction as s_t. We detail our attentive module in the next Section.

Dynamic Convolutional Filters. Dynamic filters are different from traditional, fixed filters typically used in CNN, as they depend on an input rather than being purely learnable parameters. In our case, we can think about them as specialized feature extractors reflecting the semantics of the natural language specification. For example, starting from an instruction like *head towards the red chair* our model can learn specific filters to focus on concepts such as *red* and *chair*. In this way, our model can rely on a large ensemble of specialized kernels and apply only the most suitable ones, depending on the current goal. Naturally, this approach is more efficient and flexible than learning a fixed set of filters for all the navigation steps. We use the representation of the current piece of instruction s_t to generate multiple 1×1 dynamic convolutional kernels, according to the following equation:

$$f_t = \ell_2[\tanh(W_f s_t + b_f)], \tag{6.3}$$

where $\ell_2[\cdot]$ indicates L2 normalization, and f_t is a tensor of filters reshaped to have the same number of channels as the image feature map. We then perform the dynamic convolution over the image features I_t, thus obtaining a response map for the current timestep as follows:

$$D_t = f_t * I_t. \tag{6.4}$$

As the aforementioned operation is equivalent to a dot product, we can conceive the dynamic convolution as a specialized form of dot-product attention, in which I_t acts as key and the filters in f_t act as time-varying queries. Following this interpretation, we rescale D_t by $\sqrt{d_f}$, where d_f is the dynamic filter size [164] to maintain dot products smaller in magnitude.

Action Selection. We use the response maps dynamically generated as input for the policy network. We implement it with an LSTM whose hidden state at time step t is employed to obtain the action scores. Formally:

$$h_t = \text{LSTM}([\tilde{D}_t, a_{t-1}], h_{t-1}), \quad p_t = \text{softmax}(W_a h_t + b_a), \tag{6.5}$$

where $[\cdot, \cdot]$ indicates concatenation, a_{t-1} is the one-hot encoding of the action performed at the previous timestep, and $\tilde{D}_t$ is the flattened tensor obtained from D_t. To select the next action a_t, we sample from a multinomial distribution parametrized with the output probability distribution during training, and select $a_t = \arg\max p_t$ during the test. In line with previous work, we find out that sampling during the training phase encourages exploration and improves overall performances.

Note that, as previously stated, we do not employ a high-level action space, where the agent selects the next viewpoint in the image feature map, but instead make the agent responsible of learning the sequence of low-level actions needed to perform the navigation. The agent can additionally send a specific stop signal when it considers the goal reached, as suggested by recent standardization attempts [3].

6.3.3 Encoder-Decoder Attention

The navigation instructions are very complex, as they involve not only different actions but also temporal dependencies between them. Moreover, their high average length represents an additional challenge for traditional embedding methods. For these reasons, we enrich our architecture with a mechanism to attend different locations of the sentence representation, as the navigation moves towards the goal.

In line with previous work on VLN [6, 56], we employ an attention mechanism to identify the most relevant parts of the navigation instruction. We employ the hidden state of our policy LSTM to get the information about our progress in the navigation episode and extract a time-varying query $q_t = W_q h_{t-1} + b_q$. We then project our sentence embedding into a lower dimensional space to obtain key vectors, and perform a scaled dot-product attention [164] among them:

$$\alpha_t = \frac{q_t K^T}{\sqrt{d_{att}}}, \qquad K = W_k X + b_k. \tag{6.6}$$

After a softmax layer, we obtain the current instruction embedding s_t by matrix multiplication between the initial sentence embedding and the softmax scores:

$$s_t = \text{softmax}(\alpha_t) X. \tag{6.7}$$

At each timestep of the navigation process s_t is obtained by attending the instruction embedding at different locations. The same vector is in turn used to obtain a time-varying query for attending spatial locations in the visual input.

6.3.4 Training

Our training sample consists of a batch of navigation instructions and the corresponding ground truth paths coming from the R2R (*Room-to-Room*) dataset [6] (described in Sec. 6.4). The path denotes a list of discretized viewpoints that the agent has to traverse to progress towards the goal. The agent spawns in the first viewpoint, and its goal is to reach the last viewpoint in the ground truth list. At each step, the simulator is responsible for providing the next ground truth action in the low-level action space that enables the agent to progress. Specifically, the ground truth action is computed by comparing the coordinates of the next target node in the navigation graph with the agent position and orientation. At each time step t, we minimize the following objective function:

$$L = -\sum_t y_t \log p_t, \tag{6.8}$$

where p_t is the output of our network, and y_t is the ground truth low-level action provided by the simulator at time step t. We train our network with a batch size of 128 and use Adam optimizer [90] with a learning rate of 10^{-3}. We adopt early stopping to terminate the training if the mean success rate does not improve for 10 epochs.

	Validation-Seen				Validation-Unseen			
Method	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**
Random agent	9.45	15.9	21.4	-	9.23	16.3	22.0	-
Traditional convolution [6]	6.01	38.6	52.9	-	7.81	21.8	28.4	-
Ours w/o encoder-decoder attention	5.86	41.3	51.2	36.3	7.72	22.0	29.3	19.3
Ours w/o pre-trained embedding	5.62	42.0	54.0	36.3	7.32	25.8	33.3	22.1
Ours w/ dynamic filters	**4.68**	**53.1**	**66.1**	**46.0**	**6.65**	**31.6**	**43.6**	**26.8**

Table 6.1: Ablation study for our architecture on the validation sets of R2R. The full model works better than when attention is removed or when conventional filters are used.

6.4 Experiments and Results

6.4.1 Experimental Settings

For our experiments, we employ the R2R (*Room-to-Room*) dataset [6]. This challenging benchmark builds upon Matterport3D dataset of spaces [31] and contains $7,189$ different navigation paths in 90 different scenes. For each route, the dataset provides 3 natural language instructions, for a total of 21,567 instructions with an average length of 29 words. The R2R dataset is split into 4 partitions: training, validation on seen environments, validation on unseen scenes, and test on unseen environments.

Evaluation Metrics. We adopt the same evaluation metrics employed by previous work on the R2R dataset: navigation error (NE), oracle success rate (OSR), success rate (SR), and success rate weighted by path length (SPL). NE is the mean distance in meters between the final position and the goal. SR is fraction of episodes terminated within no more than 3 meters from the goal position. OSR is the success rate that the agent would have achieved if it received an oracle stop signal in the closest point to the goal along its navigation. SPL is the success rate weighted by normalized inverse path length and penalizes overlong navigations.

Implementation Details. For each LSTM, we set the hidden size to 512. Word embeddings are obtained with GloVe [131]. In our visual encoder, we apply a bottleneck layer to reduce the dimension of the image feature map to 512. We generate dynamic filters with 512 channels using a linear layer with dropout [152] ($p = 0.5$). In our attention module, q and K have 128 channels and we apply a ReLU non-linearity after the linear transformation. For our action selection, we apply dropout with $p = 0.5$ to the policy hidden state before feeding it to the linear layer.

6.4.2 Ablation Study

In our ablation study, we test the influence of our implementation choices on VLN. As a first step, we discuss the impact of dynamic convolution by comparing our model with a similar *seq2seq* architecture that employs fixed convolutional filters. We then detail the importance of using an attention mechanism to extract the current piece of instruction to be fulfilled. Finally, we compare the results obtained using a pre-trained word embedding instead of learning the word representation from scratch. Results are reported in Table 6.1.

Static Filters Vs. Dynamic Convolution. As results show, dynamic convolutional filters surpass traditional fixed filters for VLN. This because they can easily adapt to new instructions and reflect the variability of the task. When compared to a baseline model that employs traditional convolution [6], our method performs 14.5% and 9.8% better, in terms of success rate, on the val-seen and val-unseen splits respectively.

Fixed Instruction Representation Vs. Attention. The navigation instructions are very complex and rich. When removing the attention module from our architecture, we keep the last hidden state h_N as instruction representation for the whole episode. Even with this limitation, dynamic filters achieve better results than static convolution, as the success rate is higher for both of the validation splits. Our attention module further increases the success rate by 11.8% and 9.6%.

Word Embedding from Scratch Vs. Pre-trained Embedding. Learning a meaningful word embedding is nontrivial and requires a large corpus of natural language descriptions. For this reason, we adopt a pre-trained word embedding to encode single words in our instructions. We then run the same model while trying to learn the word embedding from scratch. We discover that a pre-trained word embedding significantly eases VLN. Our model with GloVe [131] obtains 11.1% and 5.8% more on the val-seen and val-unseen splits respectively, in terms of success rate.

6.4.3 Multi-headed Dynamic Convolution

In this experiment, we test the impact of using a different number of dynamically-generated filters. We test our architecture when using 1, 2, 4, 8, and 16 dynamic filters. We find out that the best setup corresponds to the use of 4 different convolutional filters. Results in Fig. 6.3 and Table 6.2 show that the success rate and the SPL increase linearly with the number of dynamic kernels for a small number of filters, reaching a maximum at 4. The metrics then decrease when adding new parameters to the network. This suggests that a low number of dynamic filters can represent a wide variety of natural language specifications. However, as the number of dynamic filters increase, the representation provided by the convolution becomes less efficient.

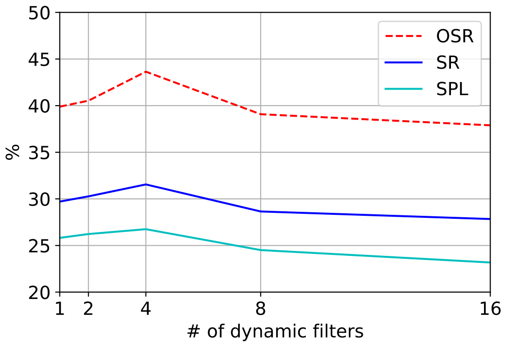


Figure 6.3: Comparison of success-based metrics using different numbers of dynamic filters on the validation-unseen set of R2R.

	Validation-Unseen			
#	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**
1	6.79	29.7	39.9	25.8
2	6.77	30.3	40.5	26.2
4	**6.65**	**31.6**	**43.6**	**26.8**
8	7.19	28.7	39.1	24.5
16	7.03	27.8	37.9	23.2

Table 6.2: Numerical comparison of the results on the main metrics using different numbers of dynamic filters. The best results for all the metrics are obtained using four different dynamic filters.

6.4.4 Comparison with the State-of-the-art

Finally, we compare our architecture with the state-of-the-art methods for VLN. Results are reported in Table 6.3. We distinguish two main categories of models, depending on their output space: the first, to which our approach belongs, predicts the next atomic action (*e.g. turn right*, *go ahead*). We call architectures in this category *low-level actions methods*. The second, instead, searches in the visual space to match the current instruction with the most suitable navigable viewpoint. In these models, atomic actions are not considered, as the agent displacements are done with a teleport system, using the next viewpoint identifier as target destination. Hence, we refer to these works as *high-level actions methods*. While the latter achieve better results, they make strong assumptions on the underlying simulating platform and on the navigation graph. Our method, exploiting dynamic convolutional filters and predicting atomic actions, outperforms comparable archi-

	Validation-Seen				**Validation-Unseen**				**Test (Unseen)**			
Low-level Methods	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**
Random	9.45	0.16	0.21	-	9.23	0.16	0.22	-	9.77	0.13	0.18	0.12
Student-forcing [6]	6.01	0.39	0.53	-	7.81	0.22	0.28	-	7.85	0.20	0.27	0.18
RPA [169]	5.56	0.43	0.53	-	7.65	0.25	0.32	-	7.53	0.25	0.33	0.23
Ours	4.68	0.53	0.66	0.46	6.65	0.32	0.44	0.27	7.14	0.31	0.42	0.27
Ours w/ data aug.	**3.96**	**0.58**	**0.73**	**0.51**	**6.52**	**0.34**	**0.43**	**0.29**	**6.55**	**0.35**	**0.45**	**0.31**

	Validation-Seen				**Validation-Unseen**				**Test (Unseen)**			
High-level Methods	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**
Speaker-Follower [56]	3.36	0.66	0.74	-	6.62	0.36	0.45	-	6.62	0.35	0.44	0.28
Self-Monitoring [109]	**3.22**	0.67	**0.78**	0.58	5.52	0.45	0.56	0.32	5.99	0.43	0.55	0.32
Regretful [110]	3.23	**0.69**	0.77	**0.63**	**5.32**	**0.50**	**0.59**	**0.41**	**5.69**	**0.48**	**0.56**	**0.40**
RCM [168]	3.37	0.67	0.77	-	5.88	0.43	0.52	-	6.12	0.43	0.50	0.38

Table 6.3: Comparison with state-of-art methods for VLN. The results for high-level models include data augmentation with synthetic data provided by [56], as in our final setup. Our method outperforms comparable models by a large margin.

tectures and achieves state of the art results for *low-level actions* VLN. Our final implementation takes advantage of the synthetic data provided by Fried *et al.* [56] and overcomes comparable methods [6, 169] by 15% and 10% success rate points on the R2R test set. Additionally, we note that our method is competitive with some *high-level actions* models, especially in terms of SPL. When considering the test set, we notice in fact that our model outperforms Speaker-Follower [56] by 3%, while performing only 1% worse than [109].

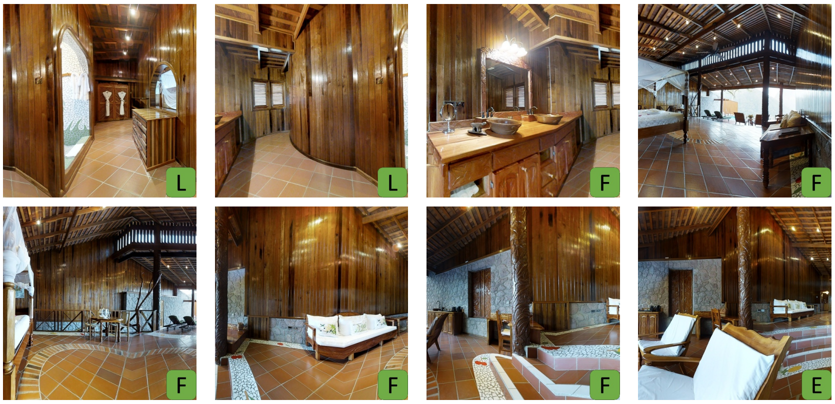
Low-level Action Space or High-level Navigation Space. While previous work on VLN never considered this important difference, we claim that it is imperative to categorize navigation architectures depending on their output space. In our opinion, ignoring this aspect would lead to inappropriate comparisons and wrong conclusions. Considering the results in Table 6.3, we separate the two classes of work and highlight the best results for each category.

Please note that the random baseline was initially provided by [6] and belongs to *low-level actions* architectures (a random *high-level actions* agent was never provided by previous work). We immediately notice that, with this new categorization, intra-class results have less variance and are much more aligned to each other. We believe that future work on VLN should consider this new taxonomy in order to provide meaningful and fair comparisons.

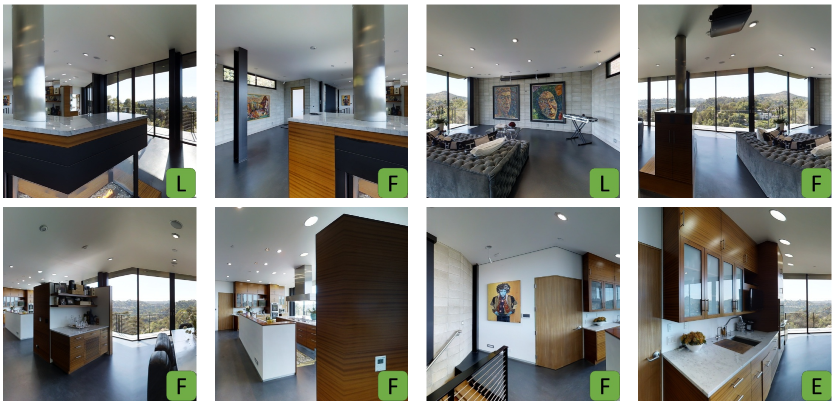
6.4.5 Qualitative Results

Fig. 6.4 shows two navigation episodes from the R2R validation set. We display the predicted action in a green box on the bottom-right corner of each image.

Legend: L *left* R *right* F *forward* E *end episode*



Instruction: *From bathroom, enter bedroom and walk straight across down two steps, wait at loungers.*



Instruction: *Walk past the fireplace and to the left. Stop in the entryway of the kitchen.*

Figure 6.4: Qualitative results from the R2R validation set. Each episode is detailed by eight pictures, representing the current position of the agent and containing the next predicted action (from left to right, top to bottom). To make the visualization more readable, we do not display the 360° panoramic images.

Both examples are successful. As we can see, our agent is able to identify and ground concepts such as *loungers*, *fireplace* and *kitchen*. These remarkable results demonstrate the flexibility and the efficacy of dynamic filters for VLN.

6.5 Conclusion

In this Chapter, we propose dynamic convolution for embodied Vision-and-Language Navigation. Instead of relying on a high-level action space, where the agent is teleported from one viewpoint to the other, we predict a series of action in an agent friendly action space. Basing on this substantial difference, we propose a new categorization based on the model output space. We then separate previous VLN architectures into *low-level actions* and *high-level actions* methods. We claim that comparisons made considering this new taxonomy are more fair and reasonable than previous analysis. Our method with dynamic convolutional filters achieves state-of-the-art results for the *low-level actions* category, and it is competitive with *high-level actions* architectures that rely on much more information and have a higher level of abstraction during the navigation episode. We hope this work encourages further research on low-level VLN, and in general we consider this a step towards the use of more realistic action spaces for this task. While our experiments show promising results in this setting, much work remains to inspect the possible connections between low-level and high-level Vision-and-Language Navigation.

Chapter 7

Perceive, Transform, and Act: VLN with Transformers

Vision-and-Language Navigation (VLN) is a challenging task in which an agent needs to follow a language-specified path to reach a target destination. Getting to the goal gets even harder as the actions available to the agent get simpler and move towards low-level, atomic interactions with the environment. This setting takes the name of low-level VLN. In this Chapter, we strive for the creation of an agent able to tackle three key issues: multi-modality, long-term dependencies, and adaptability towards different locomotive settings. To that end, we devise "Perceive, Transform, and Act" (PTA): a fully-attentive VLN architecture that leaves the recurrent approach behind and the first Transformer-like architecture incorporating three different modalities – natural language, images, and low-level actions for the agent control. In particular, we adopt an early fusion strategy to merge lingual and visual information efficiently in our encoder. We then propose to refine the decoding phase with a late fusion extension between the agent's history of actions and the perceptual modalities. We experimentally validate our model on two datasets: PTA achieves state-of-the-art results in low-level VLN on R2R and in the recently proposed R4R benchmark.

This Chapter is related to publication [7], as reported in the List of Publications.

7.1 Introduction

Effective instruction-following and contextual decision-making can open the door to a new world for researchers in Embodied AI. Deep neural networks have the potential to build complex reasoning rules that enable the creation of intelligent agents, and research on this subject could also help to empower the next generation of collaborative robots [145, 175]. In this scenario, Vision-and-Language Navigation (VLN) [6] plays a significant part in current research. This task requires to follow natural language instructions through unknown environments, discovering the correspondences between lingual and visual perception step by step. Additionally, the agent needs to progressively adjust navigation in light of the history of past actions and explored areas. Even a small error while planning the next move can lead to failure because perception and actions are unavoidably entangled; indeed, *we must perceive in order to move, but we must also move in order to perceive* [62]. For this reason, the agent can succeed in this task only by efficiently combining the three modalities – language, vision, and actions.

In Chapter 6, we identify two main operating settings for VLN [96], called *high-level action space* and *low-level action space* (Fig. 7.1). The concept of a high-level, *panoramic* action space was first proposed by Fried *et al.* [56]. In this setting, navigation takes place on a graph whose connectivity is known a priori and the nodes are represented by different viewpoints (*i.e.* the locations where the agent can step and look at the surroundings). High-level agents predict the path to the goal as a sequence of connected viewpoints, and move through the environment using a teleporting system. This aspect limits adaptability to real-world applications and prevents current research on high-level VLN from having a practical impact on embodied navigation robots. Instead, low-level methods make predictions over the agent locomotor system, hence performing actions with a one-to-one correspondence with the robot control system – *rotate* $X°$, *tilt up/down*, and *step forward* are examples of low-level actions. Even though low-level navigation can still be performed on a graph-like environment (with viewpoints as nodes), the agent is not aware of it and does not exploit any knowledge related to the structure of the underlying simulating platform. This setting is more in line with recent research on Embodied AI platforms [145, 175], which is moving towards realistic and low-level interactions with the environment and continuous control of the agent. Since the adaptability to real-world applications represents an important challenge in this scenario, we tackle the task of low-level VLN, in which abstract reasoning (such as teleporting from a viewpoint to the next and knowledge of the connectivity graph) is no longer available to the agent.

Encouraged by the success of attention in many vision-and-language tasks [51, 107, 164], we propose a new model for low-level VLN that exploits fully-attentive networks to merge the knowledge coming from different domains. In this Chapter, we devise *Perceive, Transform, and Act* (PTA), in which the different modalities

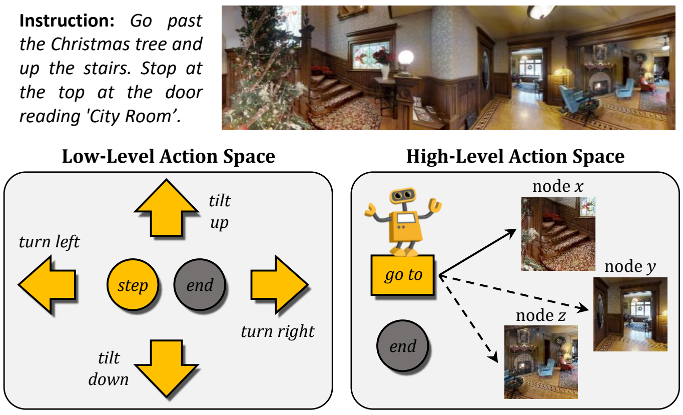

Instruction: *Go past the Christmas tree and up the stairs. Stop at the top at the door reading 'City Room'.*
Low-Level Action Space
tilt up
turn left
step
end
turn right
tilt down
High-Level Action Space
node x
go to
node y
node z
end

Figure 7.1: Previous approaches to VLN perform high-level navigation, relaxing the assumptions on the agent action space. Instead, PTA implements low-level interactions with the environment.

(text, vision, and actions) can be conditioned on the full history of previous observations. While all the previous approaches to VLN rely on a recurrent policy to track the agent's internal status through time, we directly infer the state from the observations via attention and avoid any form of recurrence (Fig. 7.2). For this reason, our agent can model the dependencies tied to navigation more efficiently and generalize to longer episodes better than other models.

At the present time, there is no study exploring the possibility for a given architecture to switch between the high-level and the low-level action spaces. In this Chapter, we experimentally show that methods born and designed for high-level navigation experience a drop in performance when adapted for low-level VLN. Indeed, high-level reasoning and abstraction from the physical environment is too heavily exploited to let the agent walk on its own. This is not true for PTA, which is designed for low-level use but can easily adapt to high-level scenarios. We summarize our main contributions as follows:

- We propose a novel multimodal framework for low-level VLN that replaces any form of recurrence with attention mechanisms, using them to tackle both long-term dependencies and multimodality. To the best of our knowledge, our model is the first Transformer-like architecture to merge visuo-linguistic perception with information coming from the agent action system;

- We technically describe how it is possible to switch from a high-level output space to a low-level locomotor system and vice versa. Experimental results on this subject are the first to analyze the mutual relationships between

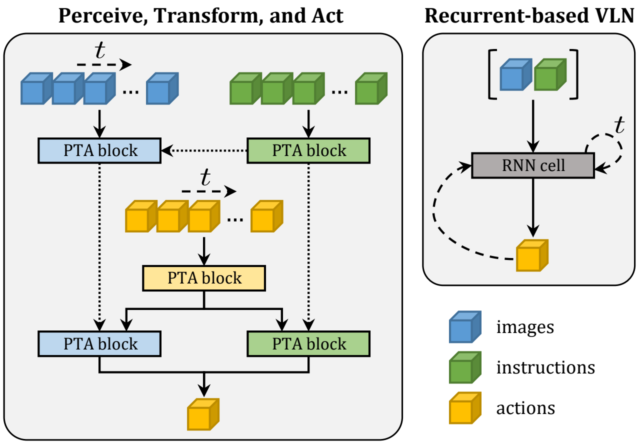


Figure 7.2: Previous architectures for VLN build upon recurrent neural networks to model long-term dependencies among the three modalities involved – text, images, and actions. Instead, PTA takes full advantage of attention mechanisms.

low-level and high-level VLN, and validate the hypothesis that high-level architectures are not easily adaptable to the low-level counterpart. Such results highlight the need for more experiments in this direction for future works;

- Experimental results show that PTA achieves state-of-the-art performance on low-level VLN. We validate this claim on two different benchmarks of increasing instruction length and complexity. Since our setting is closer to real-world applications and requires to decode fine-grained atomic actions, we believe that low-level VLN represents the next testbed for embodied agents aiming to perform Vision-and-Language Navigation.

7.2 Related Work

For a comprehensive overview of low-level and high-level Vision-and-Language Navigation, we refer the reader to Section 6.2. Here we present the related work on attention networks, as well as the challenges of long-term dependencies and multimodality implied by VLN.

Attentive Networks. The understanding and generation of language and sequences have traditionally been addressed either with recurrent [158] or convolutional [7, 59] architectures. Fully attentive models, in which recurrent relations are replaced with self and cross-attention, have recently become the dominant approach in language understanding tasks, with architectures like the Transformer [164] and BERT [51]. As a consequence, there is a growing interest in the use of fully-attentive models in visual and multimodal tasks, like video understanding [156], cross-modal retrieval [150] and image captioning [42, 104]. Our proposal is the first to employ a fully-attentive architecture for VLN and integrates vision, language, and action using cross-attention operations.

Long-term Dependency and Multimodality in VLN. Most of the difficulties and challenges of Vision-and-Language Navigation originates from long-term dependencies and multimodality. Newly proposed benchmarks for VLN such as the Room-for-Room (R4R) dataset [83] and new evaluation metrics based on Dynamic Time Warping [112] show that traditional approaches hardly adapt to longer trajectories. Indeed, the recurrent nature of previous methods exacerbates the difficulty of learning long-term dependencies [20] both in the instruction and in the navigation. In this Chapter, we employ these tools to show the effectiveness of our approach purely based on attention.

Recently, some methods propose to deal with the challenges of multimodality in VLN using a high-level perspective: a recent line of work designs graph operations to boost planning capabilities [50] or to model visuo-linguistic relationships in the graph nodes [80]. Zhang *et al.* [181] propose to employ two levels of attention-guided co-grounding, together with a new learning scheme alternating teacher-forcing and student-forcing. Qi *et al.* [132] design an architecture taking advantage from both visual tokens and action tokens in the instructions. Visual tokens are employed to identify meaningful visual features in the environment, while action tokens consider only the agent state (represented by coordinates features). In the proposed architecture, we leverage the same intuition in our multimodal decoder. In fact, we propose an additional decoding branch that does not employ visual features, but focuses on action clues provided in the sole instruction.

7.3 Proposed Method

Our goal is to navigate unseen environments using low-level actions with the only help of natural language instructions and egocentric visual observations. To merge multimodal knowledge coming from the environment, we devise a **two-stage encoder**. In the first stage, we focus on encoding the instruction – this step can be done once per episode as the natural language indication remains the same throughout the navigation. In the second stage, we use spatial attention to encode the visual observation and then employ the encoded instruction coming from the previous phase to enrich the agent representation of the surrounding environment. At each time step, the agent selects a move to progress towards the goal. To determine the next action, we fuse visuo-linguistic information with the history of

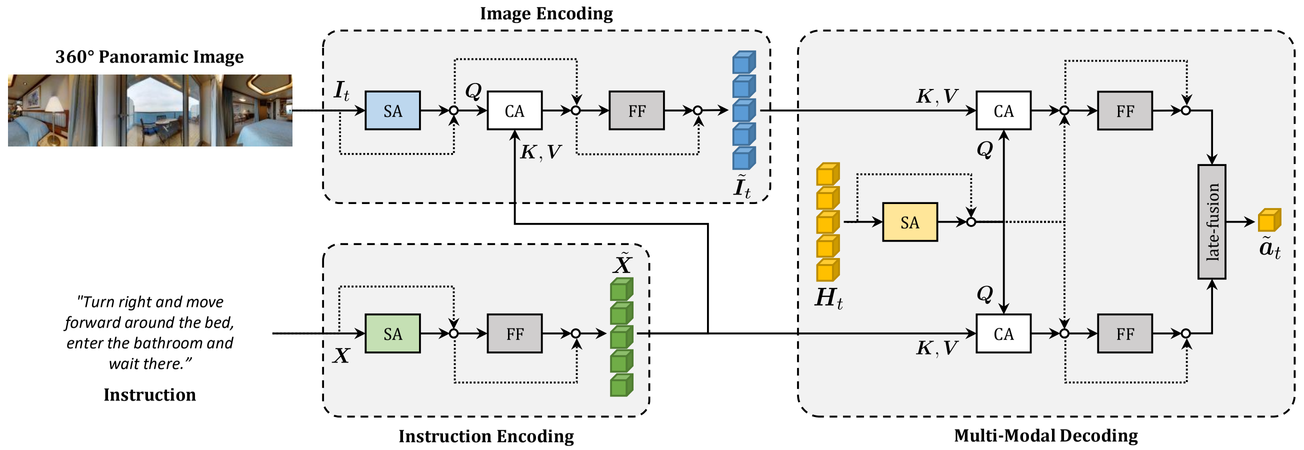


Figure 7.3: Overview of our approach. Our attention-based architecture for VLN builds upon three main blocks: an instruction encoder, an image encoder, and a multimodal decoder. SA, CA, and FF stand for self-attention, cross-attention, and feed-forward networks respectively. Dotted lines stand for residual connections between the results of the attention blocks and their inputs. For sake of clarity, we omit layer normalization after each block

actions via attention and build a **multimodal decoder** which merges the three modalities: actions, images, and text. We then decode a probability distribution over a low-level output space in which possible actions are atomic moves like *turn* or *step ahead*. After a first phase in which we train the agent with imitation learning, we implement an **extrinsic reward** function to promote coherence between ground-truth and predicted trajectories. We are the first, to the best of our knowledge, to build a VLN architecture without recurrence. Each component of our model is end-to-end trainable. Our architecture is depicted in Fig. 7.3 and detailed next.

7.3.1 Two-stage Encoder

At the beginning of each navigation episode, the agent receives a natural language instruction $\{w_0, w_1, \dots, w_{n-1}\}$ of variable length n. The agent also perceives a panoramic 360° image of the surroundings $\boldsymbol{I}_t$ at each timestep t. Our encoder consists of a single branch for each modality: text and images, and then employs attention to create a fused representation which specifically models the relevance of the source instruction into the visual observation.

Instruction Encoding. To encode the textual instruction, we employ an attention mechanism with multiple heads, followed by a feed-forward network. As a first step, we filter stop words and apply GloVe embeddings [131] to obtain a meaningful representation for each word. We then apply the following transformation:

$$\tilde{\boldsymbol{X}} = \text{LayerNorm}\left(\max(0, \boldsymbol{X}\boldsymbol{W}_x + \boldsymbol{b}_x)\right), \tag{7.1}$$

where $\boldsymbol{X}$ is the GloVe embedding for the natural language instruction, $\boldsymbol{W}_x \in \mathbb{R}^{d_{\text{GloVe}} \times d_{\text{model}}}$ and $\boldsymbol{b}_x \in \mathbb{R}^{d_{\text{model}}}$ are learnable parameters, and LayerNorm$(\cdot)$ stands for layer normalization. Since the instruction encoder has no recurrence, we must inject information about the relative position of the words in the sentence. Such information is added in the form of positional encoding to the input embeddings. The positional encodings have the same dimension as the embeddings, so that the two can be summed. We employ sine and cosine functions of different frequencies, in line with [164]:

$$\begin{aligned} PE_{(pos,2j)} &= sin(pos/10000^{2j/d_{\text{model}}}), \\ PE_{(pos,2j+1)} &= cos(pos/10000^{2j/d_{\text{model}}}), \end{aligned} \tag{7.2}$$

where *pos* is the position in the sequence and j is the channel index. At this point we use multi-head attention to create a representation that models temporal dependencies inside the instruction. Multi-head attention is defined as:

$$\begin{aligned} \text{MH}\left(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}\right) &= \text{Concat}\left(\boldsymbol{h}_1,\boldsymbol{h}_2,\dots,\boldsymbol{h}_h\right)\boldsymbol{W}^O, \\ \boldsymbol{h}_i &= \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q,\boldsymbol{K}\boldsymbol{W}_i^K,\boldsymbol{V}\boldsymbol{W}_i^V\right), \end{aligned} \tag{7.3}$$

where $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_\text{model}\times d_k}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{d_\text{model}\times d_k}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{d_\text{model}\times d_v}$, and $\boldsymbol{W}^O \in \mathbb{R}^{hd_v\times d_\text{model}}$ denote learnable weight matrices, and the index i stands for the i^th head in the multi-head attention module. As also stated in our implementation details, $d_k = d_v = d_\text{model}/h$. In each head, we employ the scaled dot-product attention defined by Vaswani *et al.* [164]:

$$\text{Attention}\left(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}\right) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}. \tag{7.4}$$

The attention mechanism described by Eq. 7.4 computes a weighted sum of the values ($\boldsymbol{V}$) basing on the similarity between the keys and the queries ($\boldsymbol{K}$ and $\boldsymbol{Q}$). In the self-attention, the same source sequence ($\tilde{\boldsymbol{X}}$ in this case) is employed to model the $(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})$ triplet of Eq. 7.3. Following the attention layer, we place a feed-forward multilayer perceptron:

$$\text{FF}\left(\tilde{\boldsymbol{X}}\right) = \max\left(0,\tilde{\boldsymbol{X}}\boldsymbol{W}_1+\boldsymbol{b}_1\right)\boldsymbol{W}_2+\boldsymbol{b}_2, \tag{7.5}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{d_\text{model}\times d_\text{ff}}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_\text{ff}\times d_\text{model}}$, $\boldsymbol{b}_1 \in \mathbb{R}^{d_\text{ff}}$, $\boldsymbol{b}_2 \in \mathbb{R}^{d_\text{model}}$. At the end of this step, we obtain the attended representation for the current instruction $\tilde{\boldsymbol{X}} = \{\tilde{\boldsymbol{x}}_0,\tilde{\boldsymbol{x}}_1,\dots,\tilde{\boldsymbol{x}}_{n-1}\}$, that we use both during image encoding and in our multimodal decoder.

Image Encoding. As a first step, we discretize the 360° panoramic image of the surroundings $\boldsymbol{I}_t$ in 36 squared locations and we extract the corresponding visual features with a ResNet-152 [73] trained on ImageNet [49]. Each viewpoint covers 30° in the equirectangular image representing the agent surroundings, hence the image representation takes the form of a 3×12 grid. We then project visual features with a transformation similar to Eq. 7.1, but instead of using sinusoidal positional encodings, we append a coordinate vector given by:

$$\text{coord}_t = \left(\sin\phi_t,\cos\phi_t,\sin\theta_t\right), \tag{7.6}$$

where $\phi_t \in (-\pi,\pi]$ and $\theta_t \in [-\frac{\pi}{2},\frac{\pi}{2}]$ are the heading and elevation angles for each viewpoint in the 3×12 grid relative to the agent position at timestep t. We then apply multi-head self-attention according to Eq. 7.3 to help modeling concepts such as relative positions between objects. In this layer, the input sequence modeling $(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V})$ is composed by the features extracted from the 36 squared regions of $\boldsymbol{I}_t$.

After this step, we aim to create an image representation enriched with the textual concepts expressed by the attended instruction $\tilde{\boldsymbol{X}}$. We use cross-attention to achieve this goal, and employ $\tilde{\boldsymbol{X}}$ as keys and values for multi-head attention (Eq. 7.3), while the queries come from the output of the previous self-attention layer. Using cross-attention, we enrich visual information with a weighted sum of the instruction tokens. From the resulting representation it is possible to draw concepts such as the *tableness* or the *redness* of an image region, given an instruction that refers to concepts such as *table* or *red*. Finally, a feed-forward network as in Eq. 7.5 is applied to obtain the attended visual observation $\tilde{\boldsymbol{I}}_t$.

7.3.2 Multimodal Decoder

Our decoder predicts the next action to perform among the following instructions: *turn right/left* 30°, *tilt up/down*, *step forward*, and *end episode* – to signal that it has reached the goal.

Contextual History for Action Decoding. The first part of our decoder takes into account the history of past actions. While previous methods employ a recurrent neural network to keep track of previous steps (see for instance [6, 109, 168]), we explicitly model $\boldsymbol{H}_t = \{a_0, a_1, \dots, a_{t-1}\}$ as the set of actions performed before the current timestep t. Note that a_0 coincides with the <*start*> token. We add sinusoidal positional encoding (Eq. 7.2) to provide temporal information and apply multi-head self-attention to obtain an attended history representation $\tilde{\boldsymbol{H}}_t = \{\tilde{\boldsymbol{a}}_0, \tilde{\boldsymbol{a}}_1, \dots, \tilde{\boldsymbol{a}}_{t-1}\}$.

Late Fusion of Perception and Action. At this point, $\tilde{\boldsymbol{H}}_t$ contains the relevant information regarding the action history of the navigation episode. However, this information must be enriched with the perception coming from the environment. We merge textual and visual information with $\tilde{\boldsymbol{H}}_t$ via attention, allowing mutual influence between perception and motion. We build two branches of multi-head cross-attention accepting respectively $\tilde{\boldsymbol{X}}$ and $\tilde{\boldsymbol{I}}_t$ as key/value pairs and using $\tilde{\boldsymbol{H}}_t$ as query. The image-action cross-attention is motivated by the fact that the agent needs to *look around* before decoding the next action. Since $\tilde{\boldsymbol{I}}_t$ already contains information coming from the instruction, this cross-attention layer is sufficient to achieve decent results on the VLN task (as demonstrated by our ablation studies). However, we find out that adding a separate text-action cross-attention layer helps generalization in unseen environments. After this step, we concatenate the two representations and apply a FC layer to obtain the output sequence whose last element corresponds to $\tilde{\boldsymbol{a}}_t$. With this last layer, we perform a late fusion of visuo-linguistic information with the agent internal state (given by its previous history). It is worth noting that PTA also comprises an early fusion mechanism: the cross-attention between $\tilde{\boldsymbol{X}}_t$ and the attended visual input introduced in the Image Encoder. In our ablation study, we discuss the positive effects given by the early fusion and the late fusion mechanisms.

Action Selection. To select the next low-level action, we project the final representation $\tilde{\boldsymbol{a}}_t$ in a six-dimensional space corresponding with the agent locomotor space containing the following actions: *turn right/left* 30°, *tilt up/down*, *step forward*, and *end episode*. The output probability distribution over the action space can therefore be written as:

$$\boldsymbol{p}_t = \text{softmax}\left(\tilde{\boldsymbol{a}}_t \boldsymbol{W}_p + \boldsymbol{b}_p\right), \tag{7.7}$$

where $\boldsymbol{W}_p \in \mathbb{R}^{d_\text{model} \times n_\text{actions}}$ and $\boldsymbol{b}_p \in \mathbb{R}^{n_\text{actions}}$ are learned parameters ($n_\text{actions} = 6$). During training, we sample the next action to perform a_t from $\boldsymbol{p}_t$, while we select $a_t = \text{argmax}(\boldsymbol{p}_t)$ during evaluation and test.

7.3.3 Training

Our training setup includes two distinct objective functions. The first estimates the policy by imitation learning, while the second enforces similarity between the ground-truth and predicted trajectories via reinforcement learning.

Imitation Learning. To approximate a good policy, we first train our agent using strong supervision. At each timestep t, the simulator outputs the ground-truth action y_t. In the low-level setup, the ground-truth action is the one that allows getting to the next target viewpoint in the minimum amount of steps. In this phase, we aim to minimize the cross-entropy loss of the predicted distribution $\boldsymbol{p}_t$ *w.r.t.* the ground-truth action y_t.

Extrinsic Reward. After a first training phase with supervised learning, we finetune our agent using an extrinsic reward function. Recently, Magalhaes *et al.* [112] propose to employ Dynamic Time Warping (DTW) [21] to evaluate the trajectories performed by navigation agents. In particular, they define the *normalized Dynamic Time Warping* (nDTW) as:

$$\text{nDTW}(R, Q) = \exp\left(-\frac{\text{DTW}(R, Q)}{|R| \cdot d_{th}}\right), \tag{7.8}$$

where R and Q are respectively the reference and the query paths, $|R|$ is the length of the reference path, and d_{th} is the success threshold distance. At each navigation step t, the agent receives a reward equal to the gain in terms of nDTW:

$$R_t = \text{nDTW}(q_{0,\dots,t}, R) - \text{nDTW}(q_{0,\dots,t-1}, R). \tag{7.9}$$

Additionally, we give an episode-level reward to the agent if it terminates the navigation within a success threshold distance d_{th} from the goal, given by $R_s =$

$\max(0, 1 - d_{goal}/d_{th})$, where d_{goal} is the final distance between the agent and the target. We can write our final reinforcement learning objective function as:

$$L_{rl} = -\mathbb{E}_{a_t \sim \pi_\theta} \left[A_t \right], \tag{7.10}$$

where the advantage function $A_t = R_t + R_s$. Based on REINFORCE algorithm [172], we derive the gradient of our reward-based objective as:

$$\nabla_\theta L_{rl} = -A_t \nabla \log \pi_\theta(a_t|s_t). \tag{7.11}$$

7.4 Low-level and High-Level Navigation

Section 7.3 describes our approach to *low-level* VLN. Here, we discuss the main technical differences with the high-level counterpart and explain how PTA can switch from one setting to the other. Differently from the low-level architectures, a high-level method aims to predict the next node to traverse in the navigation graph, as physical navigation takes place with a teleport mechanism. The choice at time step t is done with a similarity measure between the agent internal state $\boldsymbol{s}_t$ and the appearance vector for the navigable locations $\boldsymbol{v}_t$. This similarity function is normally mapped into a bilinear dot-product:

$$\boldsymbol{p}_t = \text{softmax}\left(f(\boldsymbol{s}_t)^\top g(\boldsymbol{v}_t) \right), \tag{7.12}$$

where $f(\cdot)$ and $g(\cdot)$ are generic transformations.

In principle, it is possible to substitute the final softmax classifier of a low-level architecture (Eq. 7.7) with Eq. 7.12 and change the corresponding action space. According to this observation, we can swap the action space of a model to test its adaptability to different navigation settings. While traditional approaches start from the hidden state of the recurrent policy to estimate the agent's internal state $\boldsymbol{s}_t$, we can derive it directly from $\tilde{\boldsymbol{a}}_t$:

$$\boldsymbol{s}_t = \tilde{\boldsymbol{a}}_t \boldsymbol{W}_s + \boldsymbol{b}_s, \tag{7.13}$$

where $\boldsymbol{W}_s$ and $\boldsymbol{b}_s$ are learned parameters. As $\boldsymbol{v}_t$, we select the unattended visual features augmented with the coordinate vector described by Eq. 7.6, and apply the following transformation:

$$g(\boldsymbol{v}_t) = \max\left(0, \boldsymbol{v}_t \boldsymbol{W}_v + \boldsymbol{b}_v\right), \tag{7.14}$$

where $\boldsymbol{W}_v$ and $\boldsymbol{b}_v$ are learned parameters.

In our architecture, $\tilde{\boldsymbol{a}}_t$ can fit to represent any kind of information about the current navigation. This is because it can draw knowledge from the perceptual modalities and the history of past actions directly and without the bottleneck represented by a recurrent network. Our experiments on this subject (Sec. 7.5.3) show that our model stands out from the literature in terms of adaptability. In other words, PTA can adapt to a different action space because it does not make any assumptions on the underlying simulating platform. Instead, our architecture relies on efficient visuo-linguistic fusion mechanisms designed to be agnostic towards the final action space. We will see that methods making stronger assumptions on the action space experience a larger drop in performance than PTA.

7.5 Experiments and Discussion

7.5.1 Experimental Setup

Datasets. In our experiments, we primarily test our architecture on the R2R dataset for VLN [6]. This dataset builds on the Matterport3D dataset of spaces [31], which contains complete scans of 90 different buildings. The visual data is enriched with more than 7 000 navigation paths and 21 000 natural language instructions. The episodes are divided into a training set, two validation splits (*validation-seen*, with environments that the agent has already seen during training, and *validation-unseen*, containing only unexplored buildings), and a test set. The testing phase takes place in previously unseen environments and is accessible via a test-server with a public leaderboard. While the instructions in R2R are quite long and complex (about 29 words on average), navigation episodes usually involve a limited number of steps – max 6 steps for high-level action space and max 23 steps for the low-level setup. In the R4R dataset, Jain *et al.* [83] merge the paths in R2R to create a more complex and challenging setup. Episodes become considerably longer, pushing the traditional approaches to their limits and testing their generalizability to arbitrary long instructions and more complex trajectories.

Evaluation Metrics. In line with previous literature, we mainly focus on four metrics. NE (Navigation Error) measures the mean distance from the goal and the stop point. SR (Success Rate) is the fraction of episodes concluded within a threshold distance from the target – 3 meters for all of the previous papers on the subject. OSR (Oracle SR) represents the SR that the agent would achieve if it received an oracle stop signal when passing within the threshold distance from the goal, while SPL (SR weighted by inverse Path Length) penalizes navigation episodes that deviate from the shortest path to the goal. SPL is accredited to be the most reliable metric on the R2R dataset [3], as it strongly penalizes exhaustive exploration and search methods like beam search. Recently, Jain *et al.* [83] propose to use Coverage weighted by Length Score (CLS) to replace SR for generic navigation trajectories, as this metric is also sensitive to intermediate nodes in the reference path. Additionally, Magalhaes *et al.* [112] propose Dynamic Time Warping (DTW) and derived metrics (Normalized DTW and Success weighted by normalized DTW) to measure the similarity between reference and predicted paths. These three last metrics are more meaningful on the R4R dataset than SR and SPL [83].

Implementation Details. In the instruction encoder, $d_{\text{GloVe}} = 300$. In each component of our model, we project the input features into a d_{model}-dimensional space, with $d_{\text{model}} = 512$. For multi-head attention, we employ $h = 8$ heads, thus $d_k = d_v = d_{\text{model}}/h = 64$. The internal representation of feed-forward networks has size $d_{\text{ff}} = 2048$. After each sub-module, we add a residual connection followed by layer normalization. We also apply dropout [152] with drop probability $p = 0.1$ after each linear layer. During training, we use Adam optimizer [90] with learning rate 10^{-4}, we set the batch size to 32 and reduce the learning rate by a factor 10 if the SPL on the validation unseen split does not improve for 5 consecutive epochs. We stop the training after 30 epochs without improvement on the same metric. When finetuning using REINFORCE, we set the initial learning rate to 10^{-7}.

7.5.2 Ablation Study

In our ablation study, we experimentally validate the importance of each module in our architecture. First, we ablate multimodality in our decoder. Then, we remove cross-attention between visual and lingual information in the encoder. Finally, we show the impact of using the history of actions $\boldsymbol{H}_t$ and the role of synthetic data augmentation [56] and REINFORCE. Results are shown in Table 7.1 and discussed below.

Multimodal Decoder. In our first ablation study, we use only one of the two decoder branches at the time, and we do not perform late fusion between lingual and visually-grounded information. When removing the textual branch (Table 7.1, line 3), our agent performs worse on unseen environments, hence losing potential in terms of generalization. When removing the visual modality, our PTA agent is blinded and can only count on the natural language instruction. This setup leads to success only when the instruction does not involve references to objects or visual properties of the environment – a nearly empty subset of the dataset. Indeed, the metrics for our *blind* agent are extremely low, and they do not vary between seen and unseen environments (Table 7.1, line 4). This result is meaningful in light of recent studies proving that some single-modality agents perform better than their multimodal version by removing the visual perception and overfitting on dataset biases [162].

Early Fusion of Textual and Visual Perception. As a second experiment, we remove the early fusion mechanism, namely the cross-attention layer between the textual and visual branches of our encoder, to check its contribution. If this fusion layer is redundant, we expect that the late fusion stage will compensate for the loss. Instead, we experience a drop in performance: -12% in terms of SPL in unseen

		Validation-Seen							Validation-Unseen						
#	**Method**	**NE** $\downarrow$	**SR** $\uparrow$	**OSR** $\uparrow$	**SPL** $\uparrow$	**CLS** $\uparrow$	**nDTW** $\uparrow$	**SDTW** $\uparrow$	NE $\downarrow$	**SR** $\uparrow$	**OSR** $\uparrow$	**SPL** $\uparrow$	**CLS** $\uparrow$	**nDTW** $\uparrow$	**SDTW** $\uparrow$
1	Anderson *et al.* [6]	6.01	0.39	0.53	-	-	-	-	7.81	0.22	0.28	-	-	-	-
2	PTA (pure IL, no extrinsic reward)	4.14	0.58	0.70	0.50	0.63	0.48	0.39	6.44	0.39	**0.49**	0.32	0.48	0.32	0.24
3	− multi-modal decoder (only visual)	3.90	0.61	0.72	0.54	0.65	0.52	0.44	6.56	0.36	0.46	0.29	0.47	0.32	0.22
4	− multi-modal decoder (only textual)	9.64	0.03	0.04	0.03	0.28	0.19	0.02	9.13	0.04	0.04	0.04	0.28	0.21	0.02
5	− early fusion (cross attention)	6.41	0.34	0.44	0.30	0.54	0.28	0.18	7.70	0.23	0.29	0.20	0.43	0.20	0.12
6	− action history (only last action)	5.40	0.42	0.54	0.36	0.55	0.39	0.27	7.19	0.22	0.31	0.18	0.41	0.26	0.12
7	+ data augmentation	**3.47**	**0.66**	**0.76**	0.58	0.67	0.54	0.47	**5.91**	0.40	0.48	0.34	0.50	0.36	0.25
8	+ extrinsic reward	3.58	0.65	0.74	**0.59**	**0.69**	**0.60**	**0.50**	6.00	**0.40**	0.47	**0.36**	**0.52**	**0.41**	**0.28**

Table 7.1: Ablation study proving the effectiveness of our main modules. We also show that our model can be initialized using synthethic data augmentation and then finetuned with a limited set of refined data. Adding an extrinsic reward function further improves the performance in the final model.

environments (Table 7.1, line 5). We thus prove the importance of early textual and visual fusion in our architecture for VLN.

Contextual History for Action Decoding. $\boldsymbol{H}_t$ stores past actions as a series of one hot vectors, and it is extremely helpful to model navigation history. It acts as a sort of memory for the agent, so that it knows what actions have already been made. A similar trick in LSTM-based VLN consists in adding the last action as input to the policy RNN at each step. In our model, removing $\boldsymbol{H}_t$ and using only the last action (losing all the history) causes a drop in performance: -14% and -17% on SPL and SR respectively for the Val-Unseen split (Table 7.1, line 6).

Data Augmentation. In line with previous literature, we find the use of additional synthetic instructions useful to initialize our agent. The synthetic training set was provided by Fried *et al.* [56] using a *Speaker* module. After a first training with the full set of instructions (synthetic *and* human-generated), we finetune using *only* the original R2R train set. Results are reported in Table 7.1, line 7.

Extrinsic Reward. While imitation learning allows approximating a good policy, there is still room for improvement via reinforcement learning. Wang *et al.* [168] were the first to use REINFORCE in the context of VLN to refine their navigation policy based on cross-modal matching. In line with them, we find REINFORCE beneficial for our model: our final agent sticks more closely to the reference trajectory and penalizes overlong navigations (Table 7.1, line 8).

7.5.3 Results on R2R

In our experiments on the R2R dataset [6], we test the ability of our agent to navigate unseen environments in light of previously unseen natural language instructions. The main test-bed for this experiment is represented by the R2R evaluation leaderboard, which is publicly available online.

Comparison with SOTA. In Table 7.2, we report our results on the R2R test set, together with the results achieved by other state-of-the-art architectures on VLN. Other methods that operate in the *low-level action space* are the sequence-to-sequence architecture proposed by Anderson *et al.* [6], the RPA model using a mixture of model-free and model-based reinforcement learning [169], and the recurrent architecture with dynamic convolutional filters proposed in Chapter 6. Our method overcomes the state-of-the-art on low-level VLN by a large margin (5% in terms of SPL and SR).

Although a direct comparison between the two settings is not feasible, we notice that PTA performs better than some high-level architectures in terms of SPL. Notably, we achieve this result without making any assumption on the underlying simulating platform and decoding a longer sequence of atomic moves, instead of target viewpoints. Moreover, high-level architectures can often count on efficient graph-search methods (impractical when dealing with continuous controls) to

	Test (Unseen)			
Low-level Methods	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**
Random	9.77	0.13	0.18	0.12
Anderson *et al.* [6]	7.85	0.20	0.27	0.18
Wang *et al.* [169]	7.53	0.25	0.33	0.23
Dynamic Filters [§ 6]	6.55	0.35	0.45	0.31
PTA	**6.17**	**0.40**	**0.47**	**0.36**

	Test (Unseen)			
High-level Methods	**NE ↓**	**SR ↑**	**OSR ↑**	**SPL ↑**
Fried *et al.* [56]	6.62	0.35	0.44	0.28
Ma *et al.* [109]	5.67	0.48	0.59	0.35
Wang *et al.* [168]	6.01	0.43	0.51	0.35
Ma *et al.* [110]	5.69	0.48	0.56	0.40
Ke *et al.* [88]	5.14	0.54	**0.64**	0.41
Tan *et al.* [159]	5.23	0.51	0.59	0.47
Li *et al.* [99]	**4.53**	**0.57**	0.63	**0.53**

Table 7.2: Results on the R2R test server for low-level (top) and high-level (bottom) methods. We chose the best version of each model basing on SPL.

decode the final trajectory, and on additional modules that are not present in our method. While these are effective for high-level VLN, their generalizability to a low-level setup, closer to real-world application, is yet to be tested.

Switching from Low-level to High-level. Our second experiment on R2R aims to test the effects of retraining existing models after switching their final action spaces (from high-level to low-level and vice-versa). To that end, we change the final classifier of PTA as described in Section 7.4. In this new setting, the output of the action decoder becomes a probability distribution over the adjacent nodes of the navigation graph. Once the agent decides where to go, the displacements are made automatically and there is no need to decode lower-level actions such as rotations. We train PTA from scratch in this setup, without any further hyperparameter tuning. In Table 7.3 we detail the full set of metrics obtained using PTA with the high-level classifier, and compare with the model incorporating the low-level control system. The small gap between the metrics in the two setups suggests that PTA does not take any particular advantage from the underlying action space. Of course, metrics that directly evaluate the final trajectory (like DTW-based metrics) benefits from using high-level actions with automatic oracle displacements.

In principle, every model should exhibit a decent level of elasticity towards different locomotor settings. In practice, we find out that architectural choices

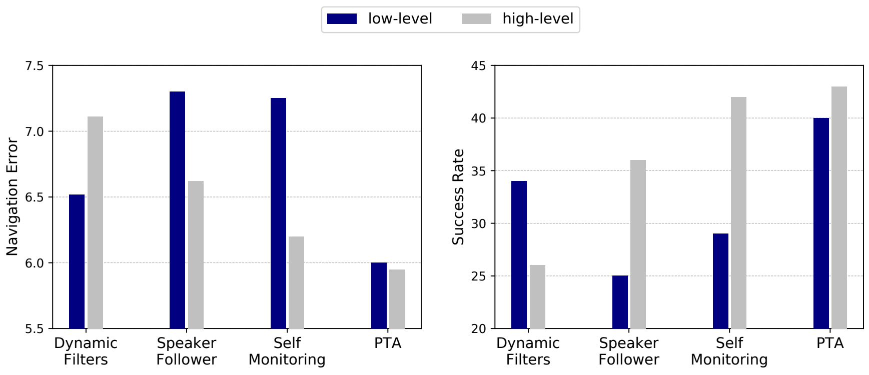


Figure 7.4: Visualization of the navigation error (left) and success rate (right) on the R2R val-unseen split. A larger difference between the blue and gray bars denotes a lower degree of adaptability. The gap is reduced when using PTA.

	R2R Validation-Unseen						
Method	**NE** ↓	**SR** ↑	**OSR** ↑	**SPL** ↑	**CLS** ↑	**nDTW** ↑	**SDTW** ↑
PTA *low-level*	6.00	0.40	0.47	0.36	0.52	0.41	0.28
PTA *high-level*	5.95	0.43	0.49	0.39	0.53	0.53	0.35

Table 7.3: Comparison between the low-level and the high-level version of PTA. On all the metrics, a small gap denotes high adaptaility. DTW-based metrics highly benefits from the use of a high-level action space.

that strongly help high-level VLN often end up hindering the other setup. This is especially true when the agent exploits high-level reasoning and makes strong assumptions on the nature of the underlying simulator. As a result, current high-level methods experience a drop in performance when adopting a simple, atomic action space (see Figure 7.4). PTA, instead, does not rely on such assumptions and builds on more efficient modules to merge multimodal information entailed in the VLN task. The plots in Figure 7.4 show that our model exhibits far greater flexibility to the final action space than other architectures. The considerably narrow step between the blue and the gray bars (representing the low-level and the high-level actions spaces respectively) denotes that a change in the final action space does not prevent PTA from reaching its goal. We compare with the Speaker-Follower [56] and the Self-Monitoring agent [109] from the high-level setup, which experience a sizeable loss in performance. In fact, results drop of 11% and 13% respectively in terms of SR when adapted for low-level use. We also compare PTA with the recurrent architecture exploiting dynamic convolution described in

Section 6.3 from the low-level category. The lower degree of adaptability shown by this method is motivated by the fact that it operates a strong compression on the visual input basing on the current instruction. In this step, much information that could ease high-level action selection is lost.

To conduct this experiment we adjust the codes from [109], which is publicly available online, and report the results in the paper for [56]. We choose the Speaker-Follower and the Self-Monitoring agents because they are flexible frameworks by design, and for this reason they are the most suitable models among their high-level peers for this comparison. We believe that the findings and insights provided in this experiment will motivate further experiments in this direction, and help to unravel the main reasons of improvements in new architectures for VLN.

Qualitative Results. In Fig. 7.5, we report a qualitative result from the R2R val-unseen set. Remarkably, PTA is able to ground concepts such as "*the second doorway on your left*" and terminates the navigation episode successfully. Since our agent operates in a low-level setup, it needs to orientate towards the next viewpoint before stepping ahead, making the decoding phase more challenging.

7.5.4 Results on R4R

R4R [83] builds upon R2R and aims to provide an even more challenging setting for embodied navigation agents. While navigation in R2R is usually direct and takes the shortest path between the starting position and the goal viewpoint, trajectories in R4R may bend and return on the agent's previous steps. This change calls for adaptation in evaluation metrics: SPL and SR are now less indicative because the agent might stop close the goal in the first half of the navigation and still fail to complete the second part. In this sense, an important role is played by recently proposed metrics: CLS [83] and nDTW [112].

In fact, CLS and nDTW take into account the agent's steps and are sensitive to intermediate errors in the navigation path. For this reason, these last metrics are more meaningful when evaluating navigation agents on R4R.

Comparison with SOTA. In this experiment, we compare PTA with other state-of-the-art architectures for VLN and report the results in Table 7.4. In the low-level setup, we compare to the recurrent architecture with dynamic convolution proposed in Section 6.3. Results show that our approach performs better on all of the main metrics. In particular, a lower NE and a higher CLS indicate that our agent tends to get closer to the goal while sticking to the natural language instruction better than the competitor. We also report the results obtained by our model incorporating the high-level decision space. We compare with Speaker-Follower [56] and RCM [168], as implemented in [83]. PTA performs better than its high-level competitors on the majority of the metrics. In particular, the higher CLS score shows that PTA can generally select a path that follows the instruction

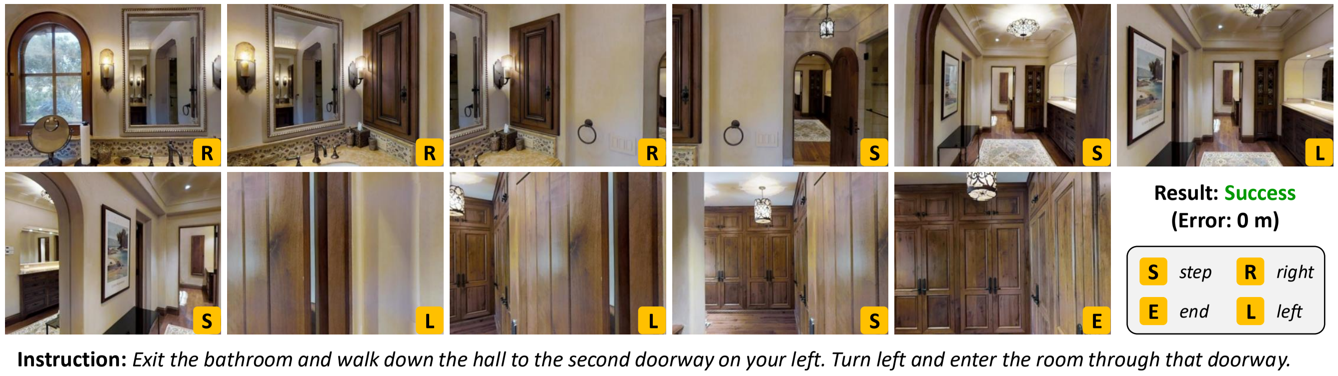

R
R
R
S
S
L
S
L
L
S
E
Result: Success
(Error: 0 m)
S step
R right
E end
L left
Instruction: Exit the bathroom and walk down the hall to the second doorway on your left. Turn left and enter the room through that doorway.

Figure 7.5: Navigation episode from the R2R unseen validation split. For each step, we report the agent first-person point of view and the next predicted action (from left to right, top to bottom).

	R4R Validation-Seen							R4R Validation-Unseen						
Method	PL ↓	NE ↓	SR ↑	SPL ↑	CLS ↑	nDTW ↑	SDTW ↑	PL ↓	NE ↓	SR ↑	SPL ↑	CLS ↑	nDTW ↑	SDTW ↑
Dynamic Filters [§ 6]	11.9	5.74	0.51	0.39	0.50	0.38	0.24	**9.98**	9.03	0.20	0.11	0.33	0.19	0.06
PTA *low-level*	**11.9**	**5.11**	**0.57**	**0.45**	**0.52**	**0.42**	**0.29**	10.2	**8.19**	**0.27**	**0.15**	**0.35**	**0.20**	**0.08**
Fried *et al.* [56]	**15.4**	5.35	0.52	0.37	0.46	-	-	19.9	8.47	0.24	**0.12**	0.30	-	-
RCM *goal oriented* [83]	24.5	5.11	0.56	0.32	0.40	-	-	32.5	8.45	**0.29**	0.10	0.20	-	-
RCM *fidelity oriented* [83]	18.8	5.37	0.53	0.31	0.55	-	-	28.5	**8.08**	0.26	0.08	0.35	-	-
PTA *high-level*	16.5	**4.54**	**0.58**	**0.39**	**0.60**	**0.58**	**0.41**	**17.7**	8.25	0.24	0.10	**0.37**	**0.32**	**0.10**

Table 7.4: Results on the R4R validation splits. Note that, since the trajectories can bind and return on the agent previous steps, CLS and nDTW are the more indicative metrics. Metrics with ‘-’ were not reported in the original papers.

better than the competitors. When considering the reference metrics proposed for R4R [83], our architecture achieves the best results on both the setups.

7.6 Conclusion

In this Chapter, we have presented *Perceive, Transform, and Act* (PTA), the first fully-attentive model for VLN. In particular, we tackle the challenging task of low-level VLN, in which high-level information about the environment is no longer accessible to the agent. We show that previous work on high-level VLN suffers from low flexibility and experiences a drop in performance when adapted for low-level use, while our agent naturally adapts to the other action space. These results suggest that boosts in performance observed in high-level VLN may be due to the use of a simpler action space, and encourage further research in this direction. Our architectural choices allow for a significant boost in performance: PTA achieves good results on low-level VLN, and when testing on the recently proposed R4R dataset, PTA achieves promising results in both the setups.

Chapter 8

Embodied Navigation in the Real World

The research field of Embodied AI has witnessed substantial progress in visual navigation and exploration thanks to powerful simulating platforms and the availability of 3D data of indoor and photorealistic environments. These two factors have opened the doors to a new generation of intelligent agents capable of achieving nearly perfect PointGoal Navigation. However, such architectures are commonly trained with millions, if not billions, of frames and tested in simulation. Together with great enthusiasm, these results yield a question: how many researchers will effectively benefit from these advances? In this Chapter, we detail how to transfer the knowledge acquired in simulation into the real world. To that end, we describe the architectural discrepancies that damage the Sim2Real adaptation ability of models trained on the Habitat simulator and propose a novel solution tailored towards the deployment in real-world scenarios. We then deploy our models on a LoCoBot, a Low-Cost Robot equipped with a single Intel RealSense camera. Different from previous work, our testing scene is unavailable to the agent in simulation. The environment is also inaccessible to the agent beforehand, so it cannot count on scene-specific semantic priors. In this way, we reproduce a setting in which a research group (potentially from other fields) needs to employ the agent visual navigation capabilities as-a-Service. Our experiments indicate that it is possible to achieve satisfying results when deploying the obtained model in the real world.

This Chapter is related to publication [6], as reported in the List of Publications.

8.1 Introduction

Embodied AI has recently attracted a lot of attention from the vision and learning communities. This ambitious research field strives for the creation of intelligent agents that can interact with the surrounding environment. Smart interactions, however, require fine-grained perception and effective planning abilities. For this reason, current research focuses on the creation of rich and complex architectures that are trained in simulation with a large amount of data. Thanks to powerful simulating platforms [48, 145, 175], the Embodied AI community could achieve nearly perfect results on the PointGoal Navigation task (PointNav) [171]. However, current research is still in the first mile of the race for the creation of intelligent and autonomous agents. Naturally, the next milestones involve bridging the gap between simulated platforms (in which the training takes place) and the real world [85]. In this Chapter, we aim to design a robot that can navigate in unknown, real-world environments.

We ask ourselves a simple research question: *can the agent transfer the skills acquired in simulation to a more realistic setting?* To answer this question, we devise a new experimental setup in which models learned in simulation are deployed on a LoCoBot [106]. Previous work on Sim2Real adaptability from the Habitat simulator [145] has focused on a setting where the real-world environment was matched with a corresponding simulated environment to test the Sim2Real metric gap. To that end, Kadian *et al.* [85] carry on a 3D acquisition of the environment specifically built for robotic experiments. Here, we assume a setting in which the final user cannot count on the technology/expertise required to make a 3D scan. This experimental setup is more challenging for the agent, as it cannot count on semantic priors on the environment acquired in simulation. Moreover, while [85] employs large boxes as obstacles, our testing scene contains real-life objects with complicated shapes such as desks, office chairs, and doors.

Our agent builds on a recent model proposed by Ramakrishnan *et al.* [135] for the PointNav task. As a first step, we research the optimal setup to train the agent in simulation. We find out that default options (tailored for simulated tasks) are not optimal for real-world deployment: for instance, the simulated agents often exploit imperfections in the simulator physics to slide along the walls. As a consequence, deployed agents tend to get stuck when trying to replicate the same sliding dynamic. By enforcing a more strict interaction with the environment, it is possible to avoid such shortcomings in the locomotor policy. Secondly, we employ the software library PyRobot [118] to create a transparent interface with the LoCoBot: thanks to PyRobot, the code used in simulation can be seamlessly deployed on the real-world agent by changing only a few lines of code. Finally, we test the navigation capabilities of the trained model on a real scene: we create a set of navigation episodes in which goals are defined using relative coordinates.

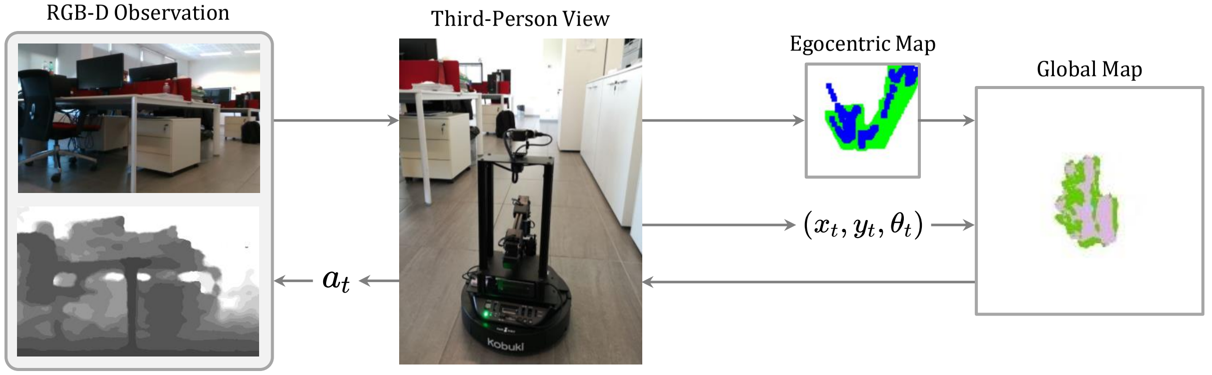


Figure 8.1: We deploy a state-of-art navigation architecture on a LoCoBot and test it in a realistic, office-like environment. Our model exploits egocentric and global occupancy maps to plan a route towards the goal.

While previous tests were mainly made in robot-friendly scenarios (often consisting of a single room), we test our model, which we call LoCoNav, in a more realistic environment with multiple rooms and typical office furniture (Fig.8.1). Thanks to our experiments, we show that models trained in simulation can adapt to real unseen environments. By making our code and models publicly available, we hope to motivate further research on Sim2Real adaptability and deployment in the real world of agents trained on the Habitat simulator.

8.2 Related Work

Embodied AI. There is a broad area of recent research that focuses on designing autonomous agents with different abilities. Among these, a vast line of work concentrates on embodied exploration and navigation [32, 36, 95, 135, 136]. In this setting, the agent's goal is to explore a new environment in the shortest amount of time. Architectures trained for this task usually employ reinforcement learning to maximize coverage (the area seen during a single episode) [32], surprisal [22], or a reward based on the novelty of explored areas [136]. Usually, this is done by creating internal map representations to keep track of the exploration progress and at the same time help the agent plan for future destinations [32, 36, 135]. The main advantage of these approaches is their ability to adapt to downstream tasks, such as PointGoal [135] or ObjectGoal [33] navigation. In PointGoal navigation, the target destination is specified using relative coordinates *w.r.t.* the agent's initial position and heading [145]. Using simulation and impressive computational power, Wijmans *et al.* [171] achieve nearly perfect results. However, their model is trained using 2.5 billion frames and requires experience acquired over more than half a year of GPU time. Unfortunately, models tend to learn simulator-specific tricks to circumvent navigation difficulties [85]. Since such shortcuts do not work in the real world, there is a significant Sim2Real performance gap.

Sim2Real Adaptation. Recent work has studied how to deploy models trained on simulation to the real world [48, 85, 141]. In their work, Kadian *et al.* [85] make a 3D acquisition of a real-world scene and study the Sim2Real gap for various setups and metrics. However, their environment is very simple as obstacles are large boxes, the floor has an even and regular surface in order to facilitate the actuation system, and there are no doors or other navigation bottlenecks. In this work, instead, we focus on a more realistic type of environment: obstacles are represented by common office furniture such as desks, chairs, cupboards; the floor is uneven as there are gaps between floor tiles that make actuation noisy and very position-dependent, and there are multiple rooms that must be accessed through doorways.

8.3 Real-World Navigation with Habitat

In this Section, we describe our out-of-the-box navigation robot. First, we describe the baseline architecture and its training procedure that takes place in the Habitat simulator [145]. Then we present our LoCoNav agent, which builds upon the baseline and implements various modules to enable real-world navigation.

8.3.1 Baseline Architecture

We draw inspiration from the occupancy anticipation agent [135] to design our baseline architecture. The model consists of three main parts: a mapper, a pose estimator, and a hierarchical policy, that we describe in the following.

Mapper. The mapper is responsible for producing an occupancy map of the environment, which is then employed by the agent as an auxiliary representation during navigation. We use two different types of map at each time step t: the agent-centric map v_t that depicts the portion of the environment immediately in front of the agent, and the global map m_t that captures the area of the environment already visited by the agent. The global map of the environment m_t is blank at $t = 0$ and it is built in an incremental way. Each map has two channels, identifying the free/occupied and the explored/unexplored space, respectively; each pixel contains the state of a 5cm $\times$ 5cm area. The mapper module takes as input the RGB and depth observations (o_t^r, o_t^d) at time t and produces the agent-centric map $v_t \in [0, 1]^{2\times V\times V}$. The RGB observation is encoded to a feature representation $\bar{o}_t^r$ with the first two layers of a pretrained ResNet-18 followed by a three-layered CNN. Instead, the depth observation is used to create a point-cloud and reprojected to form a preliminary map $\bar{o}_t^d$. The resulting agent-centric map v_t is computed by combining $\bar{o}_t^r$ and $\bar{o}_t^d$ with a U-Net. Then, v_t is registered to the global map $m_t \in [0, 1]^{2\times W\times W}$, with $W > V$, using the agent's position and heading in the environment (x_t, y_t, θ_t).

Pose Estimator. While the agent navigates towards the goal, the interactions with the environment are subject to noise and errors, so that, for instance, the action *go forward* 25*cm* might not result in a real displacement of 25cm. That could happen for a variety of reasons: bumping into an obstacle, slipping on the terrain, or simple actuation noise. The pose estimator is responsible of avoiding such positioning mistakes and keeps track of the agent pose in the environment at each time step t. This module computes the relative displacement $(\Delta x_t, \Delta y_t, \Delta\theta_t)$ caused by the action selected by the agent at time t. It takes as input the RGB-D observations (o_t^r, o_t^d) and (o_{t-1}^r, o_{t-1}^d) retrieved at time t and $t - 1$, and the egocentric maps v_t and v_{t-1}. Each modality is considered separately to obtain a first estimate of the displacement:

$$g_i = W_1 \text{max}(W_2 \star + b_2, 0) + b_1, \tag{8.1}$$

The final output of the pose estimator is the weighted sum of the three displacement vectors g_i:

$$(\Delta x_t, \Delta y_t, \Delta\theta_t) = \sum_{i=0}^{2} \alpha_i \cdot g_i, \tag{8.2}$$

$$\alpha_i = \text{softmax}(\text{MLP}_i([\bar{o}_t^r, \bar{o}_t^d, \bar{v}_t])), \tag{8.3}$$

where MLP is a three-layered fully-connected network, ($\bar{o}_t^r$, $\bar{o}_t^d$, $\bar{v}_t$) are the inputs encoded by a CNN and $[\cdot, \cdot, \cdot]$ denotes tensor concatenation. The estimated pose of the agent at time t is given by

$$(x_t, y_t, \theta_t) = (x_{t-1}, y_{t-1}, \theta_{t-1}) + (\Delta x_t, \Delta y_t, \Delta\theta_t). \tag{8.4}$$

Hierarchical Policy. Following a current trend in Embodied AI [32, 36, 135], we employ a hierarchical policy in our baseline navigator. The highest-level component of our policy is the global policy. The global policy selects a long-term goal on the global map, that we call global goal. The input of the global policy at time t is a 4-channel enriched global map $m_t^+ \in [0, 1]^{4\times W\times W}$ obtained as the concatenation of the global map m_t with a spatial representation of visited states and a one-hot representation of the agent position at time t. Finally, we compute an 8-channel input of shape $G \times G$ for the global policy. To that end, we concatenate a cropped and a max-pooled version of m_t^+. The global policy outputs a probability distribution over the $G \times G$ action space. The global goal is sampled from this distribution and is then converted to (x, y) coordinates on the global map. A new global goal is sampled every N time steps during training and is set to the navigation goal during deployment and test. The middle-level component of our hierarchical policy is the planner. After the global goal is set, an A* planner decodes the next local goal within 0.25m from the agent and on the trajectory towards the global goal. A new local goal is sampled if at least one of the following three conditions verifies: a new global goal is sampled by the global policy, the previous local goal is reached, or the local goal is known to be in an occupied area. Finally, the local policy performs the low-level navigation and decodes the series of actions to perform. The actions available to the agents are *go forwards* $25cm$ and *turn* $15°$. The local policy samples an atomic action a_t at each time step t.

8.3.2 Training in Simulation

The baseline architecture described in the previous lines is trained in simulation using Habitat [145] and 3D scans from the Gibson dataset of spaces [175]. The mapper is trained with a binary cross-entropy loss using the ground-truth occupancy maps of the environment, obtained as described in [135]. The navigation policy is trained using reinforcement learning. We choose PPO [147] as training algorithm. The global policy receives a reward signal equal to the increase in terms of anticipated map accuracy [135]:

$$R_t^{glob} = \text{Accuracy}(m_t, \hat{m}) - \text{Accuracy}(m_{t-1}, \hat{m}), \tag{8.5}$$

where m_t and m_{t-1} represent the global occupancy maps computed at time t and $t-1$ respectively, and $\hat{m} \in [0,1]^{2\times W\times W}$ is the ground-truth global map. The map accuracy is defined as:

$$\text{Accuracy}(m, \hat{m}) = \sum_{i=1}^{W^2} \sum_{j=1}^{2} \mathbb{1}[m_{ij} = \hat{m}_{ij}], \tag{8.6}$$

where $\mathbb{1}[\cdot]$ is an indicator function that returns one if the condition $[\cdot]$ is true and zero otherwise. The local policy is trained using a reward that encourages the decrease in the euclidean distance between the agent and the local goal while penalizing collisions with obstacles:

$$r_t^{local} = d_t - d_{t-1} - \alpha * \textit{bump}_t, \tag{8.7}$$

where d_t and d_{t-1} are the euclidean distances to the local goal at times t and $t-1$, $\textit{bump}_t \in \{0,1\}$ identifies a collision at time t and α regulates the contributions of the collision penalty. The training procedure described in this section exploits the experience collected throughout 6.5 million exploration frames.

8.3.3 LoCoNav: Adapting for Real World

The baseline architecture described above is trained in simulation and achieves state-of-art results on embodied exploration and navigation [135]. The reality, however, poses some major challenges that need to be addressed to achieve good real-world performances. For instance, uneven ground might give rise to errors and noise in the actuation phase. To overcome this and other discrepancies between simulated and real environments, we design LoCoNav: an agent that leverages the availability of powerful simulating platforms during training but is tailored for real-world use. In this Section, we describe the main characteristics of the LoCoNav design. We deploy our architecture on a LoCoBot [106] and use PyRobot [118] for seamless code integration.

Prevent your Agent from Learning Tricks. All simulations are imperfect. One of the main objectives when training an agent for real-world use in simulation is to prevent it from learning simulator-specific tricks instead of the basic navigation skills. During training, we observed that the agent tends to hit the obstacles instead of avoiding them. This behavior is given by the fact that the simulator allows the agent to slide towards its direction even if it is in contact with an obstacle as if there were no friction at all. Unfortunately, this ideal situation does not fit the real world, as the agent needs to actively rotate and head towards a free direction every time it bumps into an obstacle. To replicate the realistic *sticky* behavior of surfaces, we check the *bump*$_t$ flag before every step. If a collision is detected, we prevent the agent from moving forward. As a result, our final agent is more cautious about any form of collision.

Sensor and Actuation Noise. Another important discrepancy between simulation and real-world is the difference in the sensor and actuation systems. Luckily, the Habitat simulator allows for great customization of input-output dynamics, thus being very convenient for our goal. In order to train a model that is more resilient to the camera noise, we apply a Gaussian Noise Model on the RGB observations and a Redwood Noise Model [39] on the depth observations. Unfortunately, the LoCoBot RealSense camera still presents various artifacts and regions with missing depth values. For that reason, we need to restore the observation retrieved from the depth camera before using it in our architecture. To that end, we apply the hole filling algorithm described in [161], followed by the application of a median filter.

Regarding the actuation noise, we find out that the use of the incremental pose estimator (employed in the occupancy anticipation model and described in our baseline architecture) is not optimal, especially when combined with the actuation noise typical of real-world scenarios. Luckily, we can count on more precise and reliable information coming from the LoCoBot actuation system. By checking the actual rotation of each wheel at every time step, the robot can update its position step by step. We adapt the odometry sensor of the LoCoBot platform to be compliant with our architecture. To that end, the pose returned by the sensor is converted by resetting it with respect to its state at the beginning of the episode. We name $\chi_0 = (x_0, y_0, \theta_0)$ the coordinate triplet given by the odometry sensor at $t = 0$. We then define:

$$\mathbf{A} = \begin{pmatrix} \mathbf{R}_0 & \mathbf{t}_0 \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_0 & -\sin\theta_0 & x_0 \\ \sin\theta_0 & \cos\theta_0 & y_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{8.8}$$

Let us define $\mathbf{x}_t$ as the augmented position vector $(x_t, y_t, 1)$ containing the agent position at each step t. We compute the relative position of the robot as:

$$\tilde{\mathbf{x}}_t = \mathbf{A}^{-1}\mathbf{x}_t, \qquad \tilde{\theta}_t = \theta_t - \theta_0, \tag{8.9}$$

where $\tilde{\mathbf{x}}_t = (\tilde{x}_t, \tilde{y}_t, 1)$ contains the agent position after the conversion to episode coordinates. The relative position and heading is given by $\tilde{\chi}_t = (\tilde{x}_t, \tilde{y}_t, \tilde{\theta}_t)$. Note that, for $t = 0$, $\tilde{\chi}_0 = (\tilde{x}_0, \tilde{y}_0, \tilde{\theta}_0) = (0, 0, 0)$.

	Height	RGB FoV	Depth FoV	Depth Range	Obst. Height Thresh.
Default for Simulation	1.25	**H**: 90, **V**: 90	**H**: 90, **V**: 90	[0.0, 10.0]	[0.2, 1.5]
LoCoNav (ours)	0.60	**H**: 70, **V**: 90	**H**: 57, **V**: 86	[0.0, 5.00]	[0.3, 0.6]

Table 8.1: List of hyperparameters changes for Sim2Real transfer.

Hyperparameters. Finally, we noticed that typical hyperparameters employed in simulation do not match the real robot characteristics. For instance, the camera height is set to 1.25m in previous works, but the RealSense camera on the LoCoBot is placed only 0.6m from the floor. During the adaptation to the real-world robot, we change some hyperparameters to align the observation characteristics of the simulated and the real world and to match real robot constraints. These parameters are listed in Table 8.1.

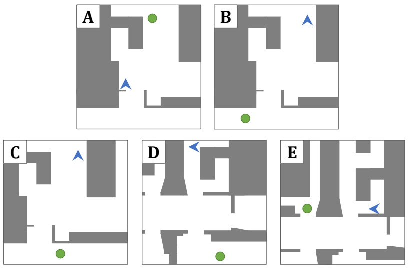
8.4 Experimental Setup

In this Section, we present our testing protocol. Experimental results obtained in a real-world environment are then presented in Section 8.5.

Testing Setup. We run multiple episodes in the real environment, in which the agent needs to navigate from a starting point A to a destination B. The goal is specified by using relative coordinates (in meters) with respect to the agent's starting position and heading. Although the agent knows the position of its destination, it has no prior knowledge of the surrounding environment. Because of this, it cannot immediately plan a direct route to the goal and must check for obstacles and walls before stepping ahead. After each run, we reset the agent memory so that it cannot retain any information from previous episodes.

We design five different navigation episodes that take place in three different office rooms and the corridor connecting them (Fig. 8.2a). For each episode, we run different trials with different configurations: obstacles are added/moved, or people are sitting/standing in the room. In total, we run 50 different experiments, resulting in more than 10 hours of real-world testing.

Evaluation Protocol. An episode is considered successful if the agent sends a specific *stop* signal within $0.2m$ from the goal. This threshold corresponds to the radius of the robot base. For every navigation episode, we also track the number of steps and the time required to reach the goal. Since the absolute number of steps is not comparable among different episodes, we ask human users to control the LoCoBot and complete each navigation path via a remote interface (we report human performance in Fig 8.2b). We then normalize these measures using this information so that results close to 1.00 indicate human-like performances. We provide absolute and normalized length and time for each episode, as well as the



(a)

Path	Length(m)	Time(s)	# Step
A	3.80	124	23
B	6.75	239	45
C	5.95	223	43
D	6.55	217	42
E	4.20	227	33

(b)

Figure 8.2: Layout of the navigation episodes (a). Path-specific information, as obtained with human supervision (b).

popular SPL metric (Success rate weighted by inverse Path Length). We employ a slightly modified version of the SPL, in which the normalization is made basing on the number of steps and not on the effective path length to penalize purposeless rotations. Additionally, we set a boolean flag for each episode that signals whether the robot has bumped into an obstacle, and we report the average Bump Rate (BR). We also report the Hard Failure Rate (HFR) as the fraction of episodes terminated if the agent gets stuck and cannot proceed, or if the episode length exceeds the limit of 300 steps.

8.5 Experimental Results

Real-world Navigation. In this experiment, we test our robot on five different realistic navigation paths (Fig. 8.2a). We report the numerical results for these experiments in Table 8.2, and we plot the main metrics in Fig. 8.3 to allow for a better visualization of navigation results across different episodes. When a path is contained in a single room (A), the agent achieves optimal results, as it always stops within the success threshold from the goal. The number of steps is slightly higher than the minimum required by the episode (33 instead of 23), but this overhead is necessary as the agent must rotate and "look around" to build a decent map of the surrounding before planning a route to the goal. Paths that involve going outside the room and navigating different spaces (B, C, D, E) are fairly complicated, but the agent can generally terminate the episode without hard failures. When the shortest path to the goal leads to a wall or a dead-end, the agent needs to find an alternative way to circumvent this obstacle (*e.g.* a door). This leads to a higher episode length because the robot must dedicate some time to general exploration of the surroundings. Finally, we find out that the most challenging scenario for our LoCoNav is when reaching the goal requires to get

Path	**SR ↑**	**SPL ↑**	**HFR ↓**	**BR ↓**	**Abs. Steps**	**Norm. Steps ↑**	**Abs. Time**	**Norm. Time ↑**
A	1.0	0.718	0.0	0.30	32.70±1.73	0.717±0.033	176.11±10.39	0.718±0.031
B	0.8	0.711	0.10	0.22	51.67±1.72	0.880±0.027	273.70±8.24	0.879±0.030
C	0.5	0.205	0.10	0.78	123.44±10.66	0.374±0.034	631.15±50.09	0.372±0.036
D	0.5	0.318	0.10	0.89	65.67±3.90	0.645±0.037	344.00±20.08	0.657±0.038
E	0.2	0.060	0.40	1.00	135.17±29.97	0.290±0.049	722.76±162.01	0.38±0.066
Overall	0.6	0.402	0.14	0.60	-	0.608±0.036	-	0.617±0.034

Table 8.2: Navigation results. Numbers after ± denote the standard error of the mean.

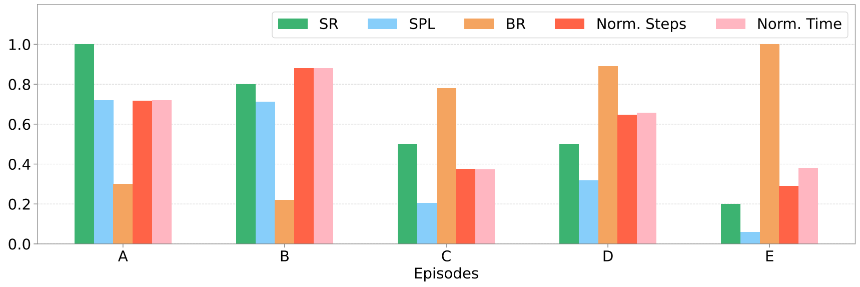


Figure 8.3: Comparison of the main navigation metrics on different episodes.

out of a room and then enter a door immediately after, on the same side of the corridor (as in E). Since the robot sticks to the shortest path, the low parallax prevents it from identifying the second door correctly. Even in these cases, a bit of general exploration helps to solve the problem.

Discussion and Failure Cases. Overall, our experimental setup provides a challenging test-bed for real-world robots. We find out that failures are due to two main issues. First, when the agent must navigate to a different room, it has no access to a map representing the general layout of the environment. This prevents the robot from computing a general plan to reach the long-term goal and forces it to explore the environment before proceeding. If a map was given to the agent, this problem would have been greatly alleviated. A second problem arises when the goal is close in terms (x, y) coordinates but is physically placed in an adjacent room. To solve this problem, one could decompose the navigation between rooms in a multi-goal problem where neighboring nodes are closer. In this way, it is possible to reduce a complex navigation episode in simpler sub-episodes (like A or B), in which our agent has proved to be successful.

8.6 Conclusion

We have presented LoCoNav, an out-of-the-box architecture for embodied navigation in the real world. Our model takes advantage of two main elements: state-of-art simulating platforms, together with a large number of 3D spaces, for efficient and fast training, and a series of techniques specifically designed for real-world deployment. Experiments are conducted in reality on challenging navigation paths and in a realistic office-like environment. Results demonstrate the validity of our approach and encourage further research in this direction.

Chapter 9

Conclusion

9.1 Summary of Contribution

This thesis contributes to the field of Embodied Artificial Intelligence. Embodied AI is a novel research topic at the intersection of Computer Vision, Natural Language Processing, and Robotics and takes advantage of recent findings on Deep Neural Networks. Empowered by the so-called Deep Revolution, we strive to create intelligent agents able to perceive the world, reason about spatio-temporal relationships, and act to reach a pre-defined goal. First, we need to identify a proper strategy to tackle such a complex topic, which entails time series and long-term dependencies on one end and multiple input modalities on the other end. We distinguish three problems we need to address to build an intelligent agent. We start from the problem of long-term dependencies and sequence modeling, as the agent needs to process data coming from a sequence of time steps acting as previous experience. Then, we consider and tackle a first simple form of interaction with an unknown environment: exploration. In this way, we combine visual and spatial reasoning to perform simple actions such as in-place rotations and moving forward. Finally, we study how to incorporate natural language instructions to guide the agent's navigation towards a goal. Language then becomes a natural interface to communicate with the agent, paving the way to future research and applications. This thesis presents a step-by-step analysis of these features that any intelligent agent should possess. While doing so, we cover a comprehensive overview of the field, theoretical foundations for Embodied AI, state-of-the-art datasets and benchmarks, and practical indications regarding the deployment of the resulting agent in the real world. Furthermore, the work presented in this thesis allows us to answer the questions raised in Chapter 1. This step-by-step

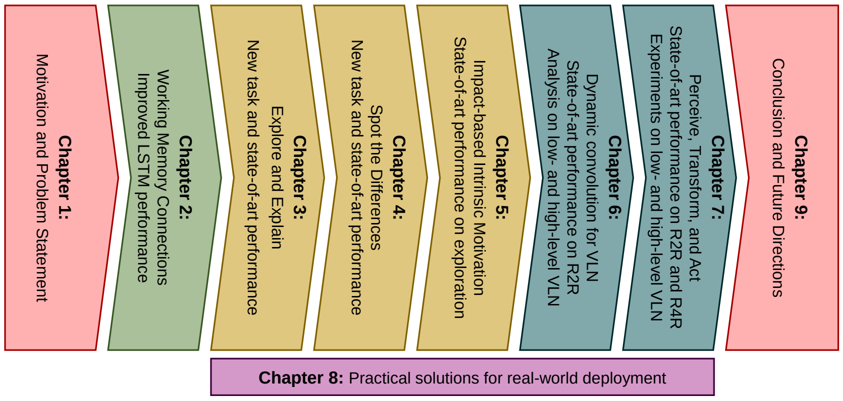

Chapter 1:
Motivation and Problem Statement
Chapter 2:
Working Memory Connections
Improved LSTM performance
Chapter 3:
Explore and Explain
New task and state-of-art performance
Chapter 4:
Spot the Differences
New task and state-of-art performance
Chapter 5:
Impact-based Intrinsic Motivation
State-of-art performance on exploration
Chapter 6:
Dynamic convolution for VLN
State-of-art performance on R2R
Analysis on low- and high-level VLN
Chapter 7:
Perceive, Transform, and Act
State-of-art performance on R2R and R4R
Experiments on low- and high-level VLN
Chapter 9:
Conclusion and Future Directions
Chapter 8: Practical solutions for real-world deployment

Figure 9.1: Summary of the work addressed in this thesis, including the major contributions and achievements.

methodology lets us move the first steps towards the new frontier of Embodied AI (see Fig. 9.1) and is the main contribution of this thesis.

In Chapter 2, we propose Working Memory Connections: a new and powerful heuristic to improve the performance of Long Short-term Memory (LSTM). Thanks to the newly-introduced design for the LSTM gates, the whole network becomes more flexible, powerful, and stable. Since long-term dependencies and time series are constant challenges in Embodied AI, we believe that the introduction of Working Memory Connections is a step towards intelligent collaborative robots.

In Chapter 3, we dive deep into embodiment. We introduce a simple yet challenging task called Explore and Explain. As the name suggests, we teach the agent to explore indoor domestic environments. We successfully achieve exploration using Deep Reinforcement Learning and a reward function based on artificial curiosity. At the same time, we design three different heuristic policies to decide when the agent is facing something worth describing. This controller activates a captioning model that produces a description of the frontal view of the robot. Enabling the agent to recount what it sees is the first step towards explainability and human-robot interactions in Embodied AI. Chapter 3 is also where we first face the challenges of multimodality in this thesis. In Explore and Explain, we choose a Divide et Impera strategy. Instead of dealing with the three modalities altogether, we model the vision-action dependencies in the navigation policy. Then, we connect vision and language using the captioner module. Last, the three modalities are united thanks to the speaker policy. As we will see in Chapters 6 and 7, a more homogeneous and unifying approach to multimodality is also possible at the cost of higher complexity.

In Chapters 4 and 5, we mainly focus on the interaction between vision and actions. We devise a novel task for embodied agents in changing environments called Spot the Difference. We draw inspiration from the fact that our homes and workplaces often change by only details. Therefore, collaborative agents must learn to deal with such changes in a map-and-update fashion. We create a new dataset and a reward function to promote this behavior in neural agents. These are the main contributions of Chapter 4. Chapter 5 concludes our exploration of embodied exploration. We devise a hierarchical model that selects long-term goals, maps the robot surroundings and plans short-horizon moves towards the goal. We train this model in simulation with deep reinforcement learning using the Habitat simulator [145] and a novel impact-based reward function. Although the impact is a purely intrinsic reward signal, the exploration abilities of our agent are unmatched. When testing on the most common setting for embodied exploration, we outperform current state-of-the-art agents trained with extrinsic rewards. Finally, we show that our modular agent can solve coordinate-based navigation thanks to the knowledge acquired during exploration training.

With this in mind, we move towards the final Chapters of the thesis, investigating the recent task of Vision-and-Language Navigation (VLN). Keeping in mind that multimodality is the main challenge of this task, we propose two flexible and elegant solutions to merge visual and textual perception. In Chapter 6, we show that a limited number of dynamically-generated convolutional kernels can ground complex concepts into the agent visual observations. Thanks to these learned correspondences, our navigation policy can decode atomic actions in a lower-dimensional space. In Chapter 7, we abandon Recurrent Neural Networks and design a fully-attentive approach to VLN. The resulting architecture jointly addresses the challenges of long-term dependencies and multimodality. Thanks to self-attention, we can model sequences and time series more effectively. On the other hand, cross-attention helps fuse information coming from different modalities. In Chapters 6 and 7, we further contribute to the field of VLN by discussing and analyzing the effects of using diverse action spaces in simulation. Recent literature, mainly driven by numerical results, had left this investigation unexplored. Throughout this thesis, technical contributions are supported by thorough experimental analysis and quantitative results on standard benchmarks and datasets.

9.2 Future Research

The step-by-step approach presented in this thesis aims to bridge the performance gap between human and robotic performance on simple Embodied AI tasks. Without any doubt, we will observe many improvements in this new and promising field in the next few years. Here, we identify three different sources of progress in Embodied AI. We envision future research as a three-lane road, as depicted in

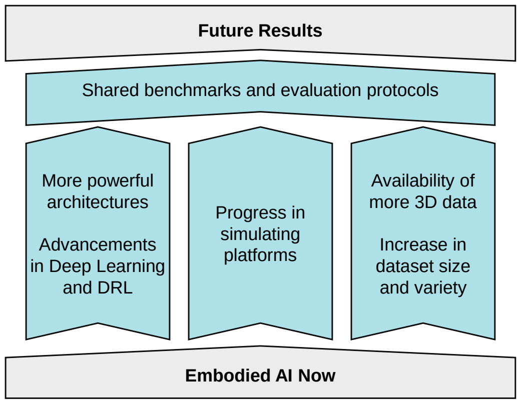

Future Results
Shared benchmarks and evaluation protocols
More powerful architectures
Advancements in Deep Learning and DRL
Progress in simulating platforms
Availability of more 3D data
Increase in dataset size and variety
Embodied AI Now

Figure 9.2: We visualize future research directions for Embodied AI as a three-lane road. To reach the long-term goal, we need progress in learning systems, simulating platforms, and availability of data. Also, standardization is a key element for progress shared among different paths.

Fig. 9.2. Each path brings advances independently from the others, yet research will need improvements in all of these topics to reach the long-term goal of intelligent collaborative agents. Let us now make these three trails for future research more precise.

Embodied Artificial Intelligence will need more powerful and robust reasoning systems to solve present and future tasks. Advances in Deep Learning and Deep Reinforcement Learning will deliver new architectures and training methodologies empowering the next generation of autonomous agents. Second, advancement in simulating platforms reproducing our everyday world will impact Embodied AI positively. Third, we will need more data for existing and future tasks to improve the generalization abilities of future robots. Orthogonally, Embodied AI will benefit from standardization and shared benchmarks and evaluation protocols to push the research community efforts in a common direction.

In recent years, we have observed a shift from Recurrent Neural Networks to Transformers in unimodal and Vision-and-Language tasks. Together with the paradigm change, results have seen a substantial improvement. We foresee that the recent novelties will empower the next generation of agents in the next future. Although the more complex setting of Embodied AI requires more time and experiments than unimodal tasks to benefit from recent discoveries, research will reach that point without any doubt. With time, the abilities of personal robots will improve thanks to future findings in Deep Learning and Deep Reinforcement Learning in the same way as present vocal assistants have benefited from recent progress. Therefore, this is the primary direction for future research.

Training agents to solve complex tasks such as Vision-and-Language Navigation would not have been possible without simulating platforms and ad hoc datasets. We envision a future in which simulators are faster, more realistic, and more flexible. Thanks to these features, we will train more agents in different settings using the same resources we now employ for a single training stage. Furthermore, realism in simulation helps deliver results in the real world. We should not forget that the deployment in our material world is the ambitious long-term goal of Embodied AI. When it comes to sim-to-real transfer, the degree of realism in the sensor and actuation systems of the simulator plays a crucial role.

Concerning existing datasets for Embodied AI, we believe that the availability of both 3D models and annotated task-related datasets play a fundamental role in future advancement. As for now, most research employs Gibson [175] and Matterport [31] datasets of 3D spaces. Recent results would not have been possible without the availability of such data. Consequently, we foresee that increase in the variety and dimension of 3D scenes will affect performance proportionally. At the same time, the mass of annotated data for Embodied AI is minimal compared to existing datasets for other settings. Although the investigation of self-supervised methods and intrinsic rewards signals is also an important direction, we expect that increasing the potential supervision will help more structured and complicated tasks. Additionally, it will yield important insights on the effects of using strong-supervised strategies to train embodied agents.

Apart from technological and methodological improvements, there will be a standardization process involving Embodied AI in the next few years. More established settings such as image captioning have already experienced this process and got strengthened by the definition of standard metrics, training procedures, and benchmarks. With this thesis, we follow the initial trails of this course.

9.3 General Conclusion

At the end of this thesis, we ask ourselves whether we have succeeded in answering our primary question, *i.e.*

How to bridge the performance gap for simple Embodied AI tasks?

The step-by-step answers presented in the various Chapters of this thesis provide an effective solution path. Since Embodied AI applications span a long-term horizon, methodologies that can solve problems with long-term dependencies are more likely to be successful. As such, mechanisms to deal with complex time series should be integrated as soon as possible to yield appropriate time modeling capabilities. Such means include gated Recurrent Neural Networks and Transformers. The embodied agent must combine these architectures with its sensory suite and actuation system to deal with multimodality inside time. A modular approach allows for a more flexible design that combines long-term planning with local decisions in time and space. Such modularity enables downstream tasks under appropriate circumstances. It is therefore favorable in general settings, opposed to ad hoc applications.

As the task becomes more complex unifying approaches get more fruitful than modular methods. Simple tools such as dynamic convolution achieve remarkable results because they address the problem efficiently. Our finding is that simple solutions are to prefer for complex tasks such as Vision-and-Language Navigation. Homogeneous solutions such as the one presented in Chapter 7 are also effective. We conclude that modular methods deal effectively with two modalities, such as vision and actions. Instead, for three and more modalities, unifying fusion techniques become fundamental.

However, current results should not lead us to excessive naive optimism. Our methods achieve state-of-the-art results on Habitat and Matterport3D benchmarks and lay the foundations for new tasks and approaches. Nevertheless, accuracy and success rates are still far from perfect. Despite the yearly progress of Embodied AI, the decisive spring towards closing the performance gap between humans and robots is still to come. Furthermore, the deployment of research results in the real world will require considerable effort. To this end, we dedicate Chapter 8 of this thesis to outline our recent results on sim-to-real adaptation. We hope our efforts in this direction will pave the way to future results.

To conclude, with the structured approach presented in this thesis, we have enriched the field of Embodied Artificial Intelligence with a practical methodology to deal with simple and complex tasks and bridge the performance gap. We are confident that an extended exploration along the path we traced, in the form of future research, will bring further development in Embodied AI.

Moving from place to place is supposed to be "physical" whereas perceiving is supposed to be "mental", but this dichotomy is misleading. Locomotion is guided by visual perception. Not only does it depend on perception but perception depends on locomotion.

The Ecological Approach to Visual Perception
James J. Gibson

Bibliography

[1] Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. In *Advances in Neural Information Processing Systems Workshops*, 2017. 32

[2] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, 2016. 26, 28

[3] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 28, 44, 46, 78, 79, 84, 104

[4] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. SPICE: Semantic Propositional Image Caption Evaluation. In *Proceedings of the European Conference on Computer Vision*, 2016. 21

[5] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 20, 26, 29, 37, 78, 80

[6] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE/CVF Conference on*

Computer Vision and Pattern Recognition, 2018. 26, 28, 29, 78, 80, 85, 86, 87, 89, 94, 101, 104, 106, 107, 108

[7] Jyoti Aneja, Aditya Deshpande, and Alexander G Schwing. Convolutional image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 96

[8] Sule Anjomshoae, Amro Najjar, Davide Calvaresi, and Kary Främling. Explainable agents and robots: Results from a systematic literature review. In *AAMAS*, 2019. 27

[9] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2015. 78, 80

[10] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2016. 18, 19, 20

[11] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3D Scene Graph: A structure for unified semantics, 3D space, and camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 49

[12] Devansh Arpit, Bhargav Kanuparthi, Giancarlo Kerg, Nan Rosemary Ke, Ioannis Mitliagkas, and Yoshua Bengio. h-detach: Modifying the LSTM Gradient Towards Better Optimization. In *Proceedings of the International Conference on Learning Representations*, 2019. 11, 19, 20

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the International Conference on Learning Representations*, 2015. 10

[14] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the Annual Meeting on Association for Computational Linguistics Workshops*, 2005. 21

[15] Lorenzo Baraldi, Costantino Grana, and Rita Cucchiara. Hierarchical boundary-aware neural encoder for video captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 10, 11

[16] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. DeepMind Lab. *arXiv preprint arXiv:1612.03801*, 2016. 29

[17] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016. 28, 60, 62, 67

[18] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *J. of Artificial Intelligence Research*, 47:253–279, 2013. 29

[19] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *Proceedings of the International Joint Conference on Neural Networks*, 1993. 10

[20] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neural Networks*, 5(2):157–166, 1994. 2, 10, 11, 97

[21] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1994. 102

[22] Roberto Bigazzi, Federico Landi, Marcella Cornia, Silvia Cascianelli, Lorenzo Baraldi, and Rita Cucchiara. Explore and Explain: Self-supervised Navigation and Recounting. In *ICPR*, 2020. 60, 116

[23] Roberto Bigazzi, Federico Landi, Marcella Cornia, Silvia Cascianelli, Lorenzo Baraldi, and Rita Cucchiara. Out of the Box: Embodied Navigation in the Real World. In *International Conference on Computer Analysis of Images and Patterns*, 2021. 60, 74

[24] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding Horizon "Next–Best–View" Planner for 3D Exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2016. 61

[25] Joydeep Biswas. The Quest For" Always-On" Autonomous Mobile Robots. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2019. 46

[26] Valentino Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT press, 1986. 2

[27] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016. 29

[28] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018. 26, 28, 31, 36

[29] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *Proceedings of the International Conference on Learning Representations*, 2018. 62

[30] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic MapNet: Building Allocentric Semantic Maps and Representations from Egocentric Views. *arXiv preprint arXiv:2010.01191*, 2020. 44, 46

[31] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. In *Proceedings of the International Conference on 3D Vision*, 2017. 34, 36, 44, 48, 53, 60, 61, 69, 86, 104, 129

[32] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning To Explore Using Active Neural SLAM. In *Proceedings of the International Conference on Learning Representations*, 2019. 44, 45, 46, 60, 61, 63, 73, 74, 75, 116, 118

[33] Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object Goal Navigation using Goal-Oriented Semantic Exploration. In *Advances in Neural Information Processing Systems*, 2020. 44, 46, 116

[34] Devendra Singh Chaplot, Lisa Lee, Ruslan Salakhutdinov, Devi Parikh, and Dhruv Batra. Embodied Multimodal Multitask Learning. *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2019. 46

[35] Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 80

[36] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning Exploration Policies for Navigation. In *Proceedings of the International Conference on Learning Representations*, 2019. 44, 46, 49, 52, 60, 62, 116, 118

[37] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv preprint arXiv:1409.1259*, 2014. 11

[38] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014. 11

[39] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015. 120

[40] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*, 2014. 11

[41] Marcella Cornia, Lorenzo Baraldi, and Rita Cucchiara. SMArT: Training Shallow Memory-aware Transformers for Robotic Explainability. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2020. 27

[42] Marcella Cornia, Matteo Stefanini, Lorenzo Baraldi, and Rita Cucchiara. Meshed-Memory Transformer for Image Captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 26, 29, 41, 97

[43] Felipe Leno Da Silva, Matthew E Taylor, and Anna Helena Reali Costa. Autonomously Reusing Knowledge in Multiagent Reinforcement Learning. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2018. 46

[44] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 78, 80

[45] Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural modular control for embodied question answering. In *Proceedings of the Conference on Robot Learning*, 2018. 78, 80

[46] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José M.F. Moura, Devi Parikh, and Dhruv Batra. Visual Dialog. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 78, 80

[47] Abhishek Das, Satwik Kottur, José M.F. Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2017. 78, 80

[48] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. RoboTHOR: An Open Simulation-to-Real Embodied AI Platform. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 60, 114, 116

[49] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2009. 83, 100

[50] Zhiwei Deng, Karthik Narasimhan, and Olga Russakovsky. Evolving graphical planner: Contextual global planning for vision-and-language navigation. In *Advances in Neural Information Processing Systems*, 2020. 97

[51] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2018. 12, 94, 97

[52] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *Proceedings of the International Conference on Learning Representations*, 2016. 63

[53] Jeffrey L. Elman. Finding Structure in Time. *Cognitive Science*, 14:179–211, 1990. 2, 10

[54] Embodied AI workshop. `https://embodied-ai.org/`. 2

[55] K Anders Ericsson and Walter Kintsch. Long-Term Working Memory. *Psychological Review*, 102(2):211, 1995. 10

[56] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems*, 2018. 78, 80, 81, 85, 89, 94, 105, 107, 108, 109, 110, 111

[57] Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. From Language to Goals: Inverse Reinforcement Learning for Vision-Based Instruction Following. In *Proceedings of the International Conference on Learning Representations*, 2019. 80

[58] Kirill Gavrilyuk, Amir Ghodrati, Zhenyang Li, and Cees GM Snoek. Actor and action video segmentation from a sentence. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 79

[59] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the International Conference on Machine Learning*, 2017. 96

[60] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Proceedings of the International Joint Conference on Neural Networks*, 2000. 10, 15, 22, 23

[61] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. 10, 12, 15

[62] James J Gibson. *The Ecological Approach to Visual Perception: Classic Edition*. Psychology Press, 2014. 2, 94

[63] Héctor H González-Banos and Jean-Claude Latombe. Navigation Strategies for Exploring Indoor Environments. *The International Journal of Robotics Research*, 21(10-11):829–848, 2002. 61

[64] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 2014. 63

[65] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in Visual Question Answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 78, 80

[66] Costantino Grana, Daniele Borghesani, and Rita Cucchiara. Optimized block-based connected components labeling with decision trees. *IEEE Transactions on Image Processing*, 19(6):1596–1609, 2010. 48

[67] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 15

[68] Alex Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2013. 10

[69] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bastiaan Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28:2222–2232, 2017. 10, 11, 13, 15

[70] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 46

[71] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 80

[72] Weituo Hao, Chunyuan Li, Xiujun Li, Lawrence Carin, and Jianfeng Gao. Towards Learning a Generic Agent for Vision-and-Language Navigation via Pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 81

[73] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016. 21, 83, 100

[74] Simao Herdade, Armin Kappeler, Kofi Boakye, and Joao Soares. Image Captioning: Transforming Objects into Words. In *Advances in Neural Information Processing Systems*, 2019. 29

[75] Rey Francis Hernandez. *Neuroethics, Nootropics, Neuroenhancement: The Ethical Case Against Pharmacological Enhancements*, volume 109. LIT Verlag Münster, 2018. 10

[76] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the International Conference on Machine Learning*, 2019. 63

[77] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München*, 1991. 10, 11

[78] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997. 10, 11, 12, 18

[79] Dirk Holz, Nicola Basilico, Francesco Amigoni, and Sven Behnke. Evaluating the Efficiency of Frontier-based Exploration Strategies. In *ISR and ROBOTIK*, 2010. 61

[80] Yicong Hong, Cristian Rodriguez-Opazo, Yuankai Qi, Qi Wu, and Stephen Gould. Language and visual entity relationship graph for agent navigation. In *Advances in Neural Information Processing Systems*, 2020. 97

[81] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational Information Maximizing Exploration. In *Advances in Neural Information Processing Systems*, 2016. 28

[82] Muhammad Zubair Irshad, Chih-Yao Ma, and Zsolt Kira. Hierarchical Cross-Modal Agent for Robotics Vision-and-Language Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2021. 66

[83] Vihan Jain, Gabriel Magalhaes, Alexander Ku, Ashish Vaswani, Eugene Ie, and Jason Baldridge. Stay on the Path: Instruction Fidelity in Vision-and-Language Navigation. In *Proceedings of Annual Meeting of the Association for Computational Linguistics*, 2019. 97, 104, 105, 110, 111, 112

[84] Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated Orthogonal Recurrent Units: On Learning to Forget. *Neural Computation*, 31(4):765–783, 2019. 10

[85] Abhishek Kadian, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra. Sim2Real Predictivity: Does evaluation in simulation predict real-world performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677, 2020. 45, 60, 74, 75, 114, 116

[86] Peter Karkus, Shaojun Cai, and David Hsu. Differentiable SLAM-net: Learning Particle SLAM for Visual Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 44, 46

[87] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015. 20, 26

[88] Liyiming Ke, Xiujun Li, Yonatan Bisk, Ari Holtzman, Zhe Gan, Jingjing Liu, Jianfeng Gao, Yejin Choi, and Siddhartha Srinivasa. Tactical Rewind: Self-Correction via Backtracking in Vision-and-Language Navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 81, 108

[89] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A doom-based ai research platform for visual reinforcement learning. In *IEEE Conf. CIG*, 2016. 29

[90] Diederik Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*, 2015. 21, 36, 37, 53, 85, 105

[91] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations*, 2013. 63

[92] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. Empowerment: A universal agent-centric measure of control. In *CEC*, 2005. 28

[93] Jacob Krantz, Erik Wijmans, Arjun Majumdar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision-and-language navigation in continuous environments. *Proceedings of the European Conference on Computer Vision*, 2020. 44

[94] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. 35

[95] Federico Landi, Lorenzo Baraldi, Marcella Cornia, Massimiliano Corsini, and Rita Cucchiara. Multimodal attention networks for low-level vision-and-language navigation. *Computer Vision and Image Understanding*, 2021. 28, 116

[96] Federico Landi, Lorenzo Baraldi, Massimiliano Corsini, and Rita Cucchiara. Embodied Vision-and-Language Navigation with Dynamic Convolutional Filters. In *Proceedings of the British Machine Vision Conference*, 2019. 94

[97] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units. *arXiv preprint arXiv:1504.00941*, 2015. 18, 19, 20

[98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 18

[99] Xiujun Li, Chunyuan Li, Qiaolin Xia, Yonatan Bisk, Asli Celikyilmaz, Jianfeng Gao, Noah Smith, and Yejin Choi. Robust Navigation with Language Pretraining and Stochastic Sampling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2019. 108

[100] Zhenyang Li, Kirill Gavrilyuk, Efstratios Gavves, Mihir Jain, and Cees G.M. Snoek. Videolstm convolves, attends and flows for action recognition. *Computer Vision and Image Understanding*, 166:41 – 50, 2018. 11

[101] Zhenyang Li, Ran Tao, Efstratios Gavves, Cees GM Snoek, and Arnold WM Smeulders. Tracking by natural language specification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 79

[102] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Proceedings of the Annual Meeting on Association for Computational Linguistics Workshops*, 2004. 21

[103] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Proceedings of the European Conference on Computer Vision*, 2014. 20, 37

[104] Guang Li Linchao Zhu Ping Liu and Yi Yang. Entangled Transformer for Image Captioning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. 97

[105] Yong Liu, Xin Hao, Biling Zhang, and Yuyan Zhang. Simplified long short-term memory model for robust and fast prediction. *Pattern Recognition Letters*, 136:81–86, 2020. 11

[106] LoCoBot: An Open Source Low Cost Robot. `https://locobot-website.netlify.com`. 6, 74, 114, 119

[107] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, 2019. 94

[108] Matteo Luperto, Michele Antonazzi, Francesco Amigoni, and N Alberto Borghese. Robot exploration of indoor environments using incomplete and inaccurate prior knowledge. *Robotics and Autonomous Systems*, 133:103622, 2020. 44

[109] Chih-Yao Ma, Jiasen Lu, Zuxuan Wu, Ghassan AlRegib, Zsolt Kira, Richard Socher, and Caiming Xiong. Self-monitoring navigation agent via auxiliary

progress estimation. In *Proceedings of the International Conference on Learning Representations*, 2019. 78, 80, 89, 101, 108, 109, 110

[110] Chih-Yao Ma, Zuxuan Wu, Ghassan AlRegib, Caiming Xiong, and Zsolt Kira. The regretful agent: Heuristic-aided navigation through progress estimation. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 78, 81, 89, 108

[111] Thi Thoa Mac, Cosmin Copot, Duc Trung Tran, and Robin De Keyser. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems*, 86:13–28, 2016. 46

[112] Gabriel Magalhaes, Vihan Jain, Alexander Ku, Eugene Ie, and Jason Baldridge. Effective and general evaluation for instruction conditioned navigation using dynamic time warping. *arXiv preprint arXiv:1907.05446*, 2019. 97, 102, 104, 110

[113] Mitchell P Marcus and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. 20

[114] Bar Mayo, Tamir Hazan, and Ayellet Tal. Visual navigation with spatial attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021. 44

[115] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*, 2018. 20

[116] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems*, 2015. 28

[117] Steven D Morad, Roberto Mecca, Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. Embodied visual navigation with automatic curriculum learning in real environments. *IEEE Robotics and Automation Letters*, 6(2):683–690, 2021. 60

[118] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta. PyRobot: An Open-source Robotics Framework for Research and Benchmarking. *arXiv preprint arXiv:1906.08236*, 2019. 74, 114, 119

[119] Lorenzo Nardi and Cyrill Stachniss. Long-term robot navigation in indoor environments estimating patterns in traversability changes. In *Proceedings*

of the IEEE International Conference on Robotics and Automation, 2020. 46

[120] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal Deep Learning. In *Proceedings of the International Conference on Machine Learning*, 2011. 2

[121] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019. 60, 61, 62

[122] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, 2016. 62, 63, 68, 70

[123] Stefan Oßwald, Maren Bennewitz, Wolfram Burgard, and Cyrill Stachniss. Speeding-Up Robot Exploration by Exploiting Background Information. *IEEE Robotics and Automation Letters*, 1(2):716–723, 2016. 60

[124] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the International Conference on Machine Learning*, 2017. 62, 67

[125] Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, 1:6, 2009. 28

[126] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, 2002. 21

[127] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *Proceedings of the International Conference on Machine Learning*, 2018. 63

[128] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*, 2013. 10

[129] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017. 26, 28, 62, 70

[130] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *Proceedings of the International Conference on Machine Learning*, 2019. 62

[131] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014. 37, 86, 87, 99

[132] Yuankai Qi, Zizheng Pan, Shengping Zhang, Anton van den Hengel, and Qi Wu. Object-and-action aware model for visual language navigation. In *Proceedings of the European Conference on Computer Vision*, 2020. 97

[133] Yuankai Qi, Qi Wu, Peter Anderson, Xin Wang, William Yang Wang, Chunhua Shen, and Anton van den Hengel. REVERIE: Remote Embodied Visual Referring Expression in Real Indoor Environments. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 80

[134] Roberta Raileanu and Tim Rocktäschel. RIDE: Rewarding impact-driven exploration for procedurally-generated environments. In *Proceedings of the International Conference on Learning Representations*, 2021. 60, 61, 62, 66, 67

[135] Santhosh K Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy Anticipation for Efficient Exploration and Navigation. In *Proceedings of the European Conference on Computer Vision*, 2020. 44, 45, 46, 52, 55, 60, 63, 65, 70, 73, 74, 114, 116, 117, 118, 119

[136] Santhosh K Ramakrishnan, Dinesh Jayaraman, and Kristen Grauman. An Exploration of Embodied Visual Exploration. *International Journal of Computer Vision*, 129(5):1616–1649, 2021. 26, 28, 44, 46, 60, 62, 116

[137] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015. 21

[138] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. 33, 37

[139] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017. 29, 37

[140] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention*, 2015. 49

[141] Marco Rosano, Antonino Furnari, Luigi Gulino, and Giovanni Maria Farinella. On Embodied Visual Navigation in Real Environments Through Habitat. In *ICPR*, 2020. 116

[142] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323:533–536, 1986. 10

[143] Muhamad Risqi U Saputra, Andrew Markham, and Niki Trigoni. Visual SLAM and structure from motion in dynamic environments: A survey. *ACM Computing Surveys*, 51(2):1–36, 2018. 46

[144] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *Proceedings of the International Conference on Learning Representations*, 2018. 46

[145] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra. Habitat: A Platform for Embodied AI Research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. 5, 26, 29, 35, 45, 53, 60, 70, 73, 80, 94, 114, 116, 117, 118, 127

[146] Jürgen Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation. *IEEE Trans. on Autonomous Mental Development*, 2(3):230–247, 2010. 28

[147] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 36, 53, 70, 119

[148] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments. *IEEE Robotics and Automation Letters*, 4(2):1699–1706, 2019. 60, 62

[149] William B. Shen, Danfei Xu, Yuke Zhu, Leonidas J. Guibas, Li Fei-Fei, and Silvio Savarese. Situational Fusion of Visual Representation for Visual Navigation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. 80

[150] Yale Song and Mohammad Soleymani. Polysemous Visual-Semantic Embedding for Cross-Modal Retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 97

[151] Mohan Sridharan and Tiago Mota. Commonsense Reasoning to Guide Deep Learning for Scene Understanding. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2020. 46

[152] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 86, 105

[153] Nitish Srivastava and Ruslan Salakhutdinov. Learning representations for multimodal data with deep belief nets. In *Proceedings of the International Conference on Machine Learning Workshops*, 2012. 3

[154] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. *Advances in Neural Information Processing Systems*, 2012. 3

[155] Cyrill Stachniss. *Robotic Mapping and Exploration*, volume 55. Springer, 2009. 60

[156] Chen Sun, Austin Myers, Carl Vondrick, Kevin Murphy, and Cordelia Schmid. VideoBERT: A Joint Model for Video and Language Representation Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. 97

[157] Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *AGI*, 2011. 28

[158] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, 2014. 10, 96

[159] Hao Tan, Licheng Yu, and Mohit Bansal. Learning to Navigate Unseen Environments: Back Translation with Environmental Dropout. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2019. 78, 81, 108

[160] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017. 28

[161] Alexandru Telea. An image inpainting technique based on the fast marching method. *J. of Graphics Tools*, 9(1):23–34, 2004. 120

[162] Jesse Thomason, Daniel Gordon, and Yonatan Bisk. Shifting the Baseline: Single Modality Performance on Visual Navigation & QA. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2018. 105

[163] Joanne Truong, Sonia Chernova, and Dhruv Batra. Bi-directional domain adaptation for sim2real transfer of embodied navigation agents. *IEEE Robotics and Automation Letters*, 6(2):2634–2641, 2021. 60

[164] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 12, 27, 29, 33, 34, 37, 41, 84, 85, 94, 97, 99, 100

[165] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. CIDEr: Consensus-based Image Description Evaluation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015. 21, 37

[166] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and Tell: A Neural Image Caption Generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2015. 10, 20, 78, 80

[167] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and Tell: Lessons Learned from the 2015 MSCOCO Image Captioning Challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):652–663, 2017. 29

[168] Xin Wang, Qiuyuan Huang, Asli Celikyilmaz, Jianfeng Gao, Dinghan Shen, Yuan-Fang Wang, William Yang Wang, and Lei Zhang. Reinforced cross-modal matching and self-supervised imitation learning for vision-language navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. 81, 89, 101, 107, 108, 110

[169] Xin Wang, Wenhan Xiong, Hongmin Wang, and William Yang Wang. Look before you leap: Bridging model-free and model-based reinforcement learning for planned-ahead vision-and-language navigation. In *Proceedings of the European Conference on Computer Vision*, 2018. 80, 89, 107, 108

[170] Saim Wani, Shivansh Patel, Unnat Jain, Angel X. Chang, and Manolis Savva. MultiON: Benchmarking Semantic Map Memory using Multi-Object Navigation. In *Advances in Neural Information Processing Systems*, 2020. 44

[171] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra. DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. In *Proceedings of the International Conference on Learning Representations*, 2020. 26, 28, 60, 73, 74, 114, 116

[172] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. 103

[173] Andrew D Wilson and Sabrina Golonka. Embodied cognition is not what you think it is. *Frontiers in psychology*, 4:58, 2013. 2

[174] Margaret Wilson. Six Views of Embodied Cognition. *Psychonomic Bulletin & Review*, 9(4):625–636, 2002. 2

[175] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 26, 29, 44, 48, 53, 56, 60, 61, 69, 70, 78, 80, 94, 114, 118, 129

[176] Huanhou Xiao, Junwei Xu, and Jinglun Shi. Exploring diverse and fine-grained caption for video by incorporating convolutional architecture into lstm-based model. *Pattern Recognition Letters*, 129:173 – 180, 2020. 11

[177] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of the International Conference on Machine Learning*, 2015. 10, 29, 78, 80

[178] Brian Yamauchi. A Frontier-Based Approach for Autonomous Exploration. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, 1997. 61

[179] Jianwei Yang, Zhile Ren, Mingze Xu, Xinlei Chen, David Crandall, Devi Parikh, and Dhruv Batra. Embodied Amodal Recognition: Learning to Move to Perceive Objects. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. 80

[180] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. Bebold: Exploration beyond the boundary of explored regions. *arXiv preprint arXiv:2012.08621*, 2020. 62

[181] Weixia Zhang, Chao Ma, Qi Wu, and Xiaokang Yang. Language-guided Navigation via Cross-Modal Grounding and Alternate Adversarial Learning. *IEEE Transactions on Circuits and Systems for Video Technology*, 2020. 97

[182] Yubo Zhang, Hao Tan, and Mohit Bansal. Diagnosing the Environment Bias in Vision-and-Language Navigation. *Proceedings of the International Joint Conferences on Artificial Intelligence*, 2020. 46

[183] Delong Zhu, Tingguang Li, Danny Ho, Chaoqun Wang, and Max Q-H Meng. Deep Reinforcement Learning Supervised Autonomous Exploration in Office Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2018. 60, 62

[184] Fengda Zhu, Yi Zhu, Xiaojun Chang, and Xiaodan Liang. Vision-language navigation with self-supervised auxiliary reasoning tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 81

[185] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017. 28

Summary

Summary (English)

This thesis contributes to the field of Embodied Artificial Intelligence. Embodied AI is a novel research topic at the intersection of Computer Vision, Natural Language Processing, and Robotics and takes advantage of recent findings on Deep Neural Networks. Empowered by the so-called ”deep revolution”, we strive to create intelligent agents able to: perceive the world, reason about spatio-temporal relationships, and act to reach a pre-defined goal. First, we need to identify a proper strategy to tackle such a complex topic, which entails time series and long-term dependencies on one end and multiple input modalities on the other end. We distinguish three different problems we need to address to build an intelligent agent. We start from the problem of long-term dependencies and sequence modeling, as the agent needs to process data coming from a sequence of time steps. Then, we consider and tackle a first simple form of interaction with an unknown environment: exploration. In this way, we combine visual and spatial reasoning to perform simple actions such as in-place rotations and moving forward. Finally, we study how to incorporate natural language instructions to guide the agent's navigation towards a goal. Language then becomes a natural interface to communicate with the agent, paving the way to future research and applications. This thesis presents a step-by-step analysis of these features that any intelligent agent should possess. While doing so, we cover a comprehensive overview of the field, theoretical foundations for Embodied AI, state-of-the-art datasets and benchmarks, and practical indications regarding the deployment of the resulting agent in the real world.

In the first part of this thesis, we discuss Recurrent Neural Networks (RNNs). RNNs are the most common approach when dealing with time series. IN particular, Long Short-Term Memory (LSTM) is the standard de-facto for many

tasks involving sequential inputs and long-term dependencies. As such, they represent an enabling technology for Embodied AI. We introduce a heuristic enhancement of LSTM that brings better results, increased training stability, and reduced convergence time on a set of tasks.

In the following, we place the agent in a simulated photorealistic unknown environment. We aim to explore the largest portion of the environment new scene in a fixed amount of time. To that end, we propose two different training setups. The first approach relies on curiosity, where the agent tries to maximize its surprisal during the exploration episode. The second strategy promotes actions likely to produce a high impact (*i.e.*visual changes) on the environment. We show that exploration is an essential ability of embodied agents and that it can enable a series of downstream tasks such as scene description and coordinate-driven navigation in unknown environments.

Finally, we tackle the recent task of Vision-and-Language Navigation (VLN). In VLN, the agent needs to follow a language-specified instruction to reach a target location in a new environment. With that in mind, we propose two different methods to fuse lingual and visual information: one based on dynamic convolutional filters and the other based on attention. This way, we show that it is possible to include natural language instructions from a human user in the agent reasoning motor. Hence, we enable a series of future research directions and applications.

As a final contribution, we discuss how to deploy agents trained in simulation in the real world. While most of our experiments exploit simulation, we show that it is possible to deploy the resulting models on a Low-Cost Robot (LoCoBot) with little effort.

Sintesi (Italiano)

Questa tesi contribuisce al campo dell'Intelligenza Artificiale Incorporata (Embodied AI). L'Embodied AI è una nuova area di ricerca all'intersezione tra visione artificiale, comprensione del linguaggio naturale e robotica e sfrutta le recenti scoperte sulle reti neurali. Il nostro obiettivo è quello di creare agenti intelligenti in grado di: percepire il mondo, ragionare sulle relazioni spazio-temporali e agire per raggiungere un obiettivo predefinito. Per affrontare questo problema, per prima cosa è necessation identifichare una strategia adeguata, poiché questo argomento complesso comporta da un lato la gestione di serie temporali con dipendenze a lungo termine, e dall'altro la presenza di input provenienti da diversi domini. Distinguiamo tre diversi problemi che dobbiamo affrontare per costruire un agente intelligente. Partiamo dal problema delle dipendenze a lungo termine e che si

Summary, in Italian.

incontrano quando è necessario processare lunghe sequenze temporali. L'agente, infatti, ha bisogno di saper elaborare dati provenienti da una serie di istanti temporali. Successivamente, consideriamo e affrontiamo una prima semplice forma di interazione con l'ambiente: l'esplorazione. In questo modo combiniamo il ragionamento visivo e spaziale per eseguire semplici azioni. Infine, studiamo come incorporare istruzioni in linguaggio naturale per guidare la navigazione dell'agente verso un obiettivo. Il linguaggio diventa quindi un'interfaccia naturale per comunicare con l'agente, aprendo le porte a ricerche e applicazioni future. In questa tesi presentiamo un'analisi di queste caratteristiche che ogni agente intelligente dovrebbe possedere. Nel fare ciò, proponiamo una panoramica completa del campo dell'Embodied AI, i suoi fondamenti teorici, i dataset e i benchmark stato dell'arte e alcune indicazioni pratiche relative all'implementazione dell'agente risultante nel mondo reale.

Nella prima parte di questa tesi, vengono discusse le Reti Neurali Ricorrenti (RNN), la tecnologia più comune per modellare serie temporali, e in particolare la Long Short-Term Memory (LSTM): lo standard di fatto per molti problemi che coinvolgono input sequenziali e dipendenze a lungo termine. In quanto tali, rappresentano una tecnologia abilitante per l'Embodied AI. Introduciamo un miglioramento euristico nella LSTM che porta a risultati migliori, maggiore stabilità durante il training e tempi di convergenza ridotti su una serie di problemi.

A seguire, collochiamo l'agente in un ambiente fotorealistico simulato. Il nostro obiettivo è esplorare più area possibile in questo nuovo ambiente in un intervallo di tempo prefissato. A tal fine, proponiamo due diverse configurazioni di training: un primo approccio basato sulla curiosità, in cui l'agente cerca di massimizzare la sua sorpresa durante l'episodio di esplorazione, e una seconda strategia basata sull'impatto dell'azione dell'agente sull'ambiente. Mostriamo che l'esplorazione è un'abilità essenziale per un agente e che può abilitare una serie di capacità più specializzate come descrivere una scena o navigare verso coordinate relative in ambienti sconosciuti.

Infine, affrontiamo il recente compito della navigazione visuale guidata da linguaggio (VLN). In questo caso l'agente deve seguire un'istruzione testuale per raggiungere la sua destinazione in un ambiente completamente nuovo. A questo fine proponiamo due diversi metodi per fondere le informazioni testuali e visive: uno basato su filtri convolutivi dinamici e l'altro basato su attenzione. In questo modo dimostriamo che è possibile includere istruzioni in linguaggio naturale provenienti da un utente umano nel motore di ragionamento dell'agente. Questa possibilità apre poi le porte una serie di future direzioni di ricerca e applicazioni.

Come contributo finale, discutiamo come portare agenti addestrati alla simulazione nel mondo reale. Mentre la maggior parte dei nostri esperimenti sfrutta la simulazione, dimostriamo che è possibile utilizzare i modelli risultanti su un Low-Cost Robot (LoCoBot) con pochi accorgimenti.

Activities Carried out During Ph.D.

Beside the research activities described in this thesis, I also took part in other teaching and service activities, which are reported below together with a list of attended conferences and schools. A complete list of publications is instead reported in the List of Publications.

Teaching Activities

- Teaching assistant for the Computer Architecture undergraduate course, University of Modena and Reggio Emilia, Academic Years 2019/20 and 2020/21

Reviewing Service

- IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)
- IEEE International Conference on Robotics and Automation (ICRA)
- IEEE Robotics and Automation Letters (RAL)
- IEEE International Conference on Pattern Recognition (ICPR)
- ACM International Conference on Multimedia (ACMMM)
- ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)
- Pattern Recognition Letters (PRL)

Conferences and Schools Attended

- Advanced Course on Data Science and Machine Learning – ACDL 2020, *Remote*
- International Conference on Pattern Recognition – ICPR 2020, *remote*
- International Computer Vision Summer School – ICVSS 2019, *Scicli (RG), Italy*
- British Machine Vision Conference – BMVC 2019, *Cardiff (UK)*

Seminars and Courses Attended

- "Research in videogames: use of deep learning for saliency estimation and cheating prevention" – Dr. Iuri Frosio (NVIDIA) – June 30, 2021
- "About Time" – Prof. Arnold Smeulders (University of Amsterdam) – September 30, 2020
- "Brain-Inspired Computing: From Neuroscience to Artificial Intelligence" – October 11, 2019
- "Computational Aspects of Deep Reinforcement Learning" – Dr. Iuri Frosio (NVIDIA) – July 15, 2019
- "ICT Technology Commercialization and Business Development for Engineers" – Steven A. Gedeon – May 22-June 5, 2019
- "Computer Graphics for Cultural Heritage Applications" – Dr. Roberto Scopigno (CNR) – March 13, 2019
- "Academic English Workshop I" – Dott. Silvia Cavalieri – February 18-March 1, 2019
- "Academic English Workshop I" – Dott. Silvia Cavalieri – February 4-12, 2019

Master Thesis Co-Advising

- Matteo Fincato, "Virtual Try-On : A Deep Learning Approach for Garment Transfer with Spatial Transformations", 2020.
- Stefano Carretti "Language-driven Visual Navigation via Attention: A Deep Dive into Vision-and-Language Navigation in Continuous Environments", 2021.

Other

- In 2019, I carried on the 3D acquisition of the museum in Galleria Estense, in Modena. One year later, the virtual spaces created for research purpose allowed to offer free guided tours to schools and young students during the Covid–19 lockdown in Italy.

List of Publications

The following list of publications includes all conference papers, journal articles, and book chapters published during my Ph.D., as well as pre-prints. Content and experimental results published in some of these papers have been included in the previous chapters, with explicit permission given by the other authors.

[1] Federico Landi, Cees GM Snoek, and Rita Cucchiara. Anomaly Locality in Video Surveillance. *Preprint*, 2019.

[2] Federico Landi, Lorenzo Baraldi, Massimiliano Corsini, and Rita Cucchiara. Embodied Vision-and-Language Navigation with Dynamic Convolutional Filters. In *Proceedings of the British Machine Vision Conference*, 2019.

[3] Roberto Bigazzi, Federico Landi, Marcella Cornia, Silvia Cascianelli, Lorenzo Baraldi, and Rita Cucchiara. Explore and Explain: Self-supervised Navigation and Recounting. In *Proceedings of the International Conference on Pattern Recognition*, 2020.

[4] Matteo Fincato, Federico Landi, Marcella Cornia, Fabio Cesari, and Rita Cucchiara. VITON-GT: An Image-based Virtual Try-On Model with Geometric Transformations. In *Proceedings of the International Conference on Pattern Recognition*, 2020.

[5] Matteo Fincato, Marcella Cornia, Federico Landi, Fabio Cesari, and Rita Cucchiara. Transform, Warp, and Dress: A New Transformation-Guided Model for Virtual Try-On. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 2021.

[6] Roberto Bigazzi, Federico Landi, Marcella Cornia, Silvia Cascianelli, Lorenzo Baraldi, and Rita Cucchiara. Out of the Box: Embodied Navigation

in the Real World. In *International Conference on Computer Analysis of Images and Patterns*, 2021.

[7] Federico Landi, Lorenzo Baraldi, Marcella Cornia, Massimiliano Corsini, and Rita Cucchiara. Multimodal Attention Networks for Low-Level Vision-and-Language Navigation. *Computer Vision and Image Understanding*, 210:103255, 2021.

[8] Federico Landi, Lorenzo Baraldi, Marcella Cornia, and Rita Cucchiara. Working Memory Connections for LSTM. *Neural Networks*, 144:334–341, 2021.

[9] Roberto Bigazzi, Federico Landi, Silvia Cascianelli, Lorenzo Baraldi, Marcella Cornia, and Rita Cucchiara. Focus on Impact: Indoor Exploration with Intrinsic Motivation. *IEEE Robotics and Automation Letters*, 2022.

[10] Federico Landi, Roberto Bigazzi, Marcella Cornia, Silvia Cascianelli, Lorenzo Baraldi, and Rita Cucchiara. Spot the Difference: A Novel Task for Embodied Agents in Changing Environment. *Preprint*, 2022.