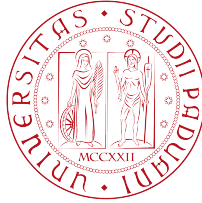


· UNIVERSITÀ DEGLI STUDI DI PADOVA ·

Department of Mathematics "Tullio Levi Civita"

Ph.D. School in Brain, Mind and Computer Science



Embodied AI with Common-Sense

Supervisors:

*Prof. Lamberto Ballan
Prof. Luciano Serafini*

Ph.D. Candidate:

Tommaso Campari

XXXV° Cycle

Abstract

Embodied AI, or artificial intelligence that is integrated into physical bodies, is an increasingly important area of research due to its potential for creating intelligent agents that can interact with and learn from the real world. These embodied agents can assist humans in a variety of tasks, such as manufacturing, healthcare, and search and rescue operations. For example, industrial robots can be equipped with artificial intelligence to assist in assembly line tasks, allowing for more efficient and accurate production. In healthcare, robots can be used to deliver medication and assist with rehabilitation exercises. In search and rescue operations, robots can be deployed to dangerous or difficult-to-reach areas to gather information and provide assistance. This thesis presents several contributions to the field of Embodied AI, addressing four research questions: (1) how can an agent exploit common-sense knowledge about the environment, (2) how can an agent reuse previously acquired knowledge about a specific environment, (3) how can an agent comply with social rules, and (4) how can an agent acquire knowledge and common-sense rules.

To address these research questions, the thesis presents a number of works that provide possible solutions. For example, one work developed a model in which a shared embedding is injected into a Scene-Memory Transformer to improve the ability of an agent to exploit common-sense knowledge about the environment. Another work defined a modular architecture for the Object Goal Navigation task that allows an agent to reuse previously acquired knowledge about a specific environment. Another work presented an agent that is able to navigate cluttered environments while being aware of social rules and the notion of risk. Finally, a preliminary end-to-end framework was presented that can simultaneously learn symbols from perceptions and symbolic functions, which could potentially be applied in an embodied agent to learn how to map perceptions to symbols and common-sense knowledge about an environment.

Overall, this thesis makes several important contributions to the field of Embodied AI, providing insights and solutions to a range of challenges faced by intelligent agents operating in the physical world.

Acknowledgements

First, I would like to thank my supervisors, Lamberto Ballan and Luciano Serafini, for the precious ideas and feedback they provided me along this journey. I want to thank Angel Chang for having hosted me under her guidance at Simon Fraser University; I will never forget the time spent there. I am really thankful to the Data and Knowledge Management (DKM) members: Gianluca Apriceno, Tommaso Carraro, Alessandro Daniele, Leonardo Lamanna, Sagar Malhotra, and Davide Rigoni. Furthermore, I am thankful to the Visual Intelligence and Machine Perception group members, particularly Enrico Cancelli and Guglielmo Camporese. I am, and always will be, enormously grateful to my mother, Irene. You are no longer here, but if I have reached this incredible milestone, I owe it to you. I am sure you've guided me through this fantastic journey, and I hope I have made you proud of me. Last but for sure not least, I am immensely grateful to my wife, family, and partner in crime, Federica. You always supported me in good and, especially, bad moments, but together we never gave up. I can not wait to see where life will take us.

Contents

1	Introduction	1
1.1	Embodied AI	1
1.2	Common-Sense Knowledge for EAI.	6
1.3	Contributions	9
2	Exploiting Scene-Specific Features for Object Goal Navigation	13
2.1	Introduction	13
2.2	Related Works	15
2.3	Dataset	17
2.4	Method	19
2.4.1	Problem Setup	19
2.4.2	Model	19
2.5	Experimental Results	22
2.5.1	Experimental setup	22
2.5.2	Results	24
2.6	Conclusion	25
3	Online Learning of Reusable Abstract Models for Object Goal Navigation	29
3.1	Introduction	29
3.2	Related Works	32
3.3	Preliminaries	34
3.4	Object Goal Navigation	34
3.5	Approach	35
3.5.1	Abstract Model Reuse	37
3.6	Implementation Details	38
3.6.1	Evaluation Metrics	38
3.7	Experiments	39
3.7.1	Reusing Abstract Models	39
3.7.2	Effects of Knowledge Accumulation	40

3.7.3	Semantic Maps and Abstract Models	41
3.7.4	Limitations and Failure analysis	43
3.7.5	Qualitative examples	44
3.8	Conclusion	46
4	Modular Multi-Object Navigation	47
4.1	Introduction	47
4.2	Related Work	49
4.3	MultiON 2.0 Dataset	51
4.4	Modular-MON	54
4.5	Experiments	57
4.5.1	Multi-Object Navigation Task	57
4.5.2	Metrics	58
4.5.3	Baselines	58
4.5.4	Results	59
4.5.5	Transferability of Modular-MON	60
4.5.6	Generalization of Modular-MON on n-ON	61
4.5.7	Object Detection on Natural objects	62
4.5.8	MultiON 2.0 distractors vs. no distractors	62
4.5.9	Qualitative Results and Analysis	63
4.5.10	Qualitative Examples	64
4.6	Conclusion	65
5	Exploiting Socially-Aware Tasks for Embodied Social Navigation	73
5.1	Introduction	73
5.2	An evaluation protocol for SocialNav	77
5.2.1	Evaluation Protocol	78
5.3	Method	80
5.3.1	Policy Architecture	81
5.3.2	Socially-Aware Tasks	82
5.3.3	Implementation details	83
5.4	Experiments	84
5.4.1	Results	85
5.4.2	Fine-grained evaluation	87
5.5	Conclusion	93
6	Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions	95
6.1	Introduction	95
6.2	Background	99

6.3	Problem Definition	100
6.3.1	Policy Functions	101
6.3.2	DSL for Direct NeSy-functions	103
6.3.3	Learning the Perception Functions	103
6.3.4	DSL for Recurrent NeSy-functions	104
6.3.5	Learning Symbolic Functions	105
6.3.6	Gradient Analysis for the Greedy Policy	106
6.4	Limitations	110
6.5	Conclusion	110
7	Conclusion	111
A	Publications	133
A.1	International Conferences and Workshops	133
A.2	ArXiv Papers	133
A.3	Under Submission	134
A.4	Extra	134

List of Figures

1.1	Passive AI tasks are based on predictions over independent samples of the world, such as images collected without a closed loop with a decision-making agent. In contrast, embodied AI tasks include an active artificial agent, such as a robot, that must perceive and interact with the environment purposely to achieve its goals, including in unstructured or even uncooperative settings. Enabled by the progress in computer vision and robotics, embodied AI represents the next frontier of challenges to study and benchmark intelligent models and algorithms for the physical world.	2
2.1	Distribution of objects in the Matterport3D Dataset.	18
2.2	SMTSC model: a) Features processing: all features are processed and a shared representation is created. This representation is then inserted into a memory (b) Scene Memory Transformer) from which the action that the agent will perform is extracted with a Scene Memory Transformer model.	20
2.3	The structure of the encoder-decoder model based on Multi-Head Attention, as presented in Fang et al.[62].	21
2.4	Successful navigation episode with SMTSC model on seen test set.	25
2.5	Unsuccessful navigation episode with SMTSC model on seen test set.	26
2.6	Successful navigation episode with SMTSC model on unseen test set.	26
2.7	Unsuccessful navigation episode with SMTSC model on unseen test set.	27

3.1	During its navigation in a complex 3D scenario, an agent incrementally acquires knowledge about the environment by storing rich semantic information in an Abstract Model. For instance, when the robot is in s_0 , chair and table are visible; by performing act_0 , other objects become visible, thus the Abstract Model is updated to s_1 . Our work shows how knowledge can be incrementally learned and effectively reused over time. . . .	30
3.2	Overview of the proposed architecture for Object Goal Navigation.	35
3.3	A plot of the moving average success rate in the ANS*+SI model. The window size was set to 5.	42
3.4	Barplot of failure cases for ANS*+SI and SemExp*+SI models on the validation set of Matterport3D.	44
3.5	A failed episode with the ANS* variant. The agent explored the environment for 500 steps without finding any Sofa occurrence. Green pixels on the map are the obstacles, light blue pixels are explored areas, and the blue point is the goal position.	45
3.6	A success with ANS*+SI variant. The agent correctly matched its current state with one in the domain and used the information stored to navigate towards a Sofa successfully.	45
4.1	Example 5ON episode on an HM3D dataset scene. The agent needs to find, in order: red, pink, black, blue, and yellow cylinders. Another three cylinders are distractors in the scene.	48
4.2	Comparing path lengths across tasks. We include boxplots for (a) average episodes length across {ObjectNav [19], 3ON [176], 3ON 2.0 (ours), and 5ON 2.0 (ours)} datasets; (b) average geodesic distance between successive objects across different datasets.	52
4.3	Modular-MON. We adopt a modular approach to multi-object navigation. The <i>Object detection</i> module transforms raw RGB to semantic labels. These are projected onto a top-down semantic map using depth observations by the <i>Map building</i> module. This serves as input for the <i>Exploration</i> module to uncover unseen areas of the environment. Finally, using exploratory or task goals, a low-level policy predicts the action for the agent to execute.	53

- 4.4 **Visualizing exploration strategies.** (*Top*) shows a *Stubborn* strategy which selects one of four corners of a local map in a clockwise direction. (*Bottom*) shows a *Random* strategy where the agent randomly samples exploration goals. We also employ corresponding variants, particularly, *Random w/ threshold* and *Stubborn w/ threshold* that have a step threshold (see Sec. 4.4). 66
- 4.5 **Natural objects.** The set of eight NAT-objects vary in shapes, sizes and colors and easily blend in the HM3D houses, thus requiring better visual understanding for the agent. 67
- 4.6 **Qualitative results.** Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and discovers objects and progressively builds the semantic map using egocentric depth observations. The goal sequence is (cyan, yellow, and finally pink). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the *found* action. 67
- 4.7 **Qualitative results: 5ON.** Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and discovers objects and progressively builds the semantic map using egocentric depth observations. The goal sequence is (black, red, yellow, pink, and finally green.). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the *found* action. The agent has a 100% Success, 100% Progress, 39% SPL and 39% PPL in this episode. 68
- 4.8 **Qualitative results: CYL objects.** Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and detects objects (‘Predicted’ column) and progressively builds the semantic map using egocentric depth observations. The goal sequence is (pink, blue, and finally green). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the *found* action. The agent has a 100% Success, 100% Progress, 21% SPL and 21% PPL in this episode. 69

4.9	Qualitative results: Natural objects. Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and discovers target objects and progressively builds the semantic map using egocentric depth observations. The goal sequence is (backpack (black), teddy bear (white), and finally trolleybag (cyan)). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the <i>found</i> action. The agent has a 100% Success, 100% Progress, 17% SPL and 17% PPL in this episode.	70
4.10	Failure cases. Most frequent failure cases include (clockwise from top left) (1) Goal has not been discovered before reaching maximum steps quota (2500); (2) PointNav fails to generate <i>stop</i> action; (3) Agent stops too far from the goal (>1meter), leading to termination of the episode; (4) All goals have been discovered but maximum steps quota has been reached before navigating to the later goals.	71
5.1	Illustration of an agent-person “encounter”. From top-left to bottom-right: <i>i</i> episode starts; <i>ii</i> the embodied agent/robot sees a person; <i>iii</i> it moves back to avoid a collision; <i>iv</i> it successfully reaches the goal by avoiding the person.	74
5.2	A scheme representing the four different classes of encounter. The dashed line represents the general direction of the agent and the person involved. The red area represents the agent’s field of view at the beginning of the encounter.	78
5.3	Pipeline and model overview. <i>Social information</i> is extracted from Habitat Simulator (left rectangle) and is processed through a <i>Social Feature extraction</i> procedure (top-right). The policy (bottom-right) uses RGB-D and GPS+Compass data as input and, during training, is conditioned by the extracted social features.	80
5.4	Implementation of a regressor network. Actions $a_t..a_{t+k}$ are used as input. A linear layer compresses the GRU’s hidden states to obtain the predicted social features $\hat{s}_t..s_{t+k}$. $\{s_i\}_{i \in [t, t+k]}$ is the ground truth used by \mathcal{L}_f (from Equation 5.3).	84
5.5	Confusion matrix obtained from the questionnaire, normalized for each true label.	87
5.6	ALV and AD curves of top models for each encounter class.	88
5.7	Number of encounters vs ESR for all the models.	89

5.8	Average Distance (on the left) and Linear Velocity curves (on the right) for the Blind-Corner class.	90
5.9	Comparison of Blind-corner ALV curves with each Socially-Aware task combination and the respective model with aux tasks added.	91
5.10	Two success cases: a frontal encounter (top) and an intersection encounter (bottom).	92
6.1	Architecture of Deep Symbolic Learning for the Sum task. Red arrows represent the backward signal during learning. . .	102
6.2	Architecture of Deep Symbolic Learning for the simple recurrent NeSy functions.	104
6.3	Tensor \mathbf{W} is used by the policy to generate tensor \mathbf{G} . This is done by applying the policy on the output dimension (vertical axes in the image), selecting a single output element for each pair of symbols $(s_1, s_2) \in \mathcal{S}_1 \times \mathcal{S}_2$	105
6.4	Confusion matrix for the MNIST digits: (left) before the permutation; (right) after permutation.	107

List of Tables

2.1	Statistics of our dataset	18
2.2	Results on the Seen Test Set	23
2.3	Results on the Unseen Test Set	23
3.1	Results achieved by our variants on the Matterport3D validation set.	41
3.2	Results obtained on a subset of the validation set of Matterport3D, containing object classes which are in both MS-COCO and MatterPort3D datasets (658 episodes across 11 environments).	41
3.3	Results obtained on the validation set of the Matterport3D dataset (2195 episodes across 11 environments). Note that SMNet (GT) , as explained in [31], exploits ground truth free space maps extracted directly from the Habitat API.	42
4.1	Comparing dataset statistics. $\#[\text{Tr}, \text{Val}, \text{Test}]$ is the number of scenes respectively in the training, validation, and test splits. $\#\text{Ep.}$ is the number of episodes in each split. We created MultiON 2.0, for 1ON to 5ON, with the same number of episodes.	51
4.2	Quantitative results (navigation policies). We find that the analytical BFS Path Planner path-planner performs considerably worse than the learned PointNav on the 3ON (cylinders) task. We employ the best-performing Random w/ threshold across these baselines.	57
4.3	Quantitative results (exploration strategies). We find a random exploration strategy with a step threshold performs better than the learned Stubborn [107] exploration strategy (and a step variant). The above experiments are on 3ON (cylinders) task. For all row entries, we employ the best-performing PointNav as Navigation module.	57

4.4	Quantitative results (OracleSem & PredictedSem). The PredictedSem agent performs significantly worse than the OracleSem agent, due to error in the Object Detector. The above set of experiments are on 3ON (cylinders) folds using the best-performing modules for Navigation (PointNav) and Exploration (Random w/ threshold).	58
4.5	Transferability. Above experiments show that the Modular-MON performs better than end-to-end models when transferred (without finetuning) to unseen environments. We observe that the OracleSem agent achieves 21% success on the MP3D scenes even when the PointNav, used as a Navigation module, is trained on HM3D scenes. On the other hand, none of the end-to-end models, trained on MP3D scenes transfers well to HM3D scenes.	60
4.6	OracleSem performance on 1ON vs 3ON vs 5ON. We observe that the agent performance deteriorates with more number of target objects.	61
4.7	PredictedSem performance on CYL-objects vs NAT-objects The PredictedSem agent with Nat-objects performs worse than the one with the CYL-objects. This shows that natural objects with varying sizes, shapes and colors are harder to detect than cylinder objects with same sizes and shapes with varying colors. The above set of experiments are on 3ON folds using PointNav as the Navigation and Random w/ threshold as the Exploration module.	62
4.8	Effect of distractors on OracleSem performance. We observe that the Modular-MON agent performs equally well in the presence of distractors. This can be attributed to our target location retrieval method from the semantic map comparing directly with the next goal category.	63
5.1	Social Navigation evaluation on Gibson4+ and HM3D-S. For each model are listed the type of input data it uses (<i>Sensors</i> column) and, eventually, what kind of self-supervised <i>Aux tasks</i> or <i>Social tasks</i> the model employs. The metrics reported are <i>Success rate</i> , <i>SPL</i> and Human-Collisions Rate (<i>H-collisions</i>).	84
6.1	Results obtained on the MNIST sum task. TE is the time required for 1 epoch, and #E the number of epochs of training. The SOTA methods are NeurASP (NAP), DeepProbLog (DPL), and DeepStochLog (DStL). SOTA results taken from [182].	108

Chapter 1

Introduction

The field of artificial intelligence has made great strides in recent years, with advances in machine learning allowing for the development of intelligent systems that can perform a wide range of tasks. However, there is still a significant gap between the abilities of these systems and those of humans, who possess a rich understanding of the world that allows them to navigate and interact with the environment in a flexible and intelligent manner. By leveraging common-sense knowledge, social-rules, and the ability to reuse previously acquire knowledge, humans have the capability to understand and reason about the world around them. We are interested in building AI agents that are capable to mimic such a behaviour. This will allow for creating AI systems that are better equipped to assist humans in their daily lives, such as reordering the house, cooking, cleaning, or grabbing specific objects in other rooms.

1.1 Embodied AI

What is Embodied AI? *Embodied AI* [58] studies artificial systems that express intelligent behavior through bodies interacting with their environments. The first generation of embodied AI researchers focused on robotic embodiments [129], arguing that robots need to interact with their noisy environments with a rich set of sensors and effectors, creating high-bandwidth interaction that breaks the fundamental assumptions of clean inputs, clean outputs, and static world states required by *classical AI* approaches [181]. More recent embodied AI research has been empowered by rich simulation frameworks, often derived from scans of real buildings and models of real robots, to recreate environments more closely resembling the real world than those previously available. These environments have enabled both discoveries

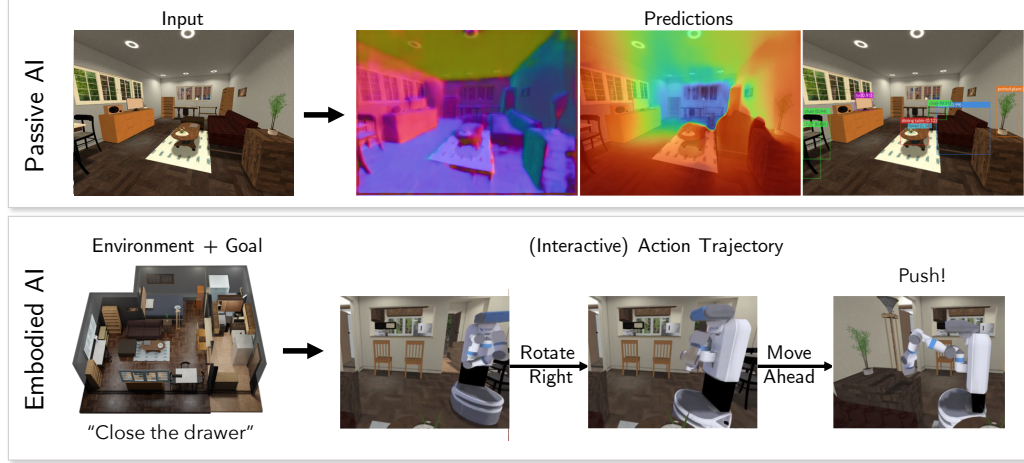


Figure 1.1: Passive AI tasks are based on predictions over independent samples of the world, such as images collected without a closed loop with a decision-making agent. In contrast, embodied AI tasks include an active artificial agent, such as a robot, that must perceive and interact with the environment purposely to achieve its goals, including in unstructured or even uncooperative settings. Enabled by the progress in computer vision and robotics, embodied AI represents the next frontier of challenges to study and benchmark intelligent models and algorithms for the physical world.

about the properties of intelligence [123] and systems which show excellent sim-to-real transfer [197, 164].

Abstracting away from real or simulated embodiments, embodied AI can be defined as the study of intelligent agents that can *see* (or more generally perceive their environment through vision, audition, or other senses), *talk* (*i.e.* hold a natural language dialog grounded in the environment), *listen* (*i.e.* understand and react to audio input anywhere in a scene.), *act* (*i.e.* navigate their environment and interact with it to accomplish goals), and *reason* (*i.e.* consider the long-term consequences of their actions). Embodied AI focuses on tasks which break the clean input/output formalism of passive tasks such as object classification and speech understanding, and require agents to interact with - and sometimes even modify - their environments over time (Fig. 1.1). Furthermore, embodied AI environments generally violate the clean dynamics of structured environments such as games and assembly lines, and require agents to cope with noisy sensors, effectors, dynamics, and other agents, which creates unpredictable outcomes.

Why is Embodiment Important? Embodied AI can be viewed as a reaction against extreme forms of the *mind-body duality* in philosophy, which some perceive to view intelligence as a purely mental phenomenon. The mind-body problem has faced philosophers and scientists for millennia [50]: humans are simultaneously “physical agents” with mass, volume, and other bodily properties, and at the same time “mental agents” that think, perceive, and reason in a conceptual domain which seems to lack physical embodiment. Some scholars argue in favor of a strict mind-body duality in which intelligence is a purely mental quality only loosely connected to bodily experience [143]. Other scholars, across philosophy, psychology, cognitive science, and artificial intelligence, have challenged this mind-body duality, arguing that intelligence is intrinsically connected to embodiment in bodily experience and that separating them has distorting effects on research [25, 127, 169, 112, 143].

The history of research in artificial intelligence has mirrored this debate over mind and body, focusing first on computational solutions for symbolic problems which appear hard to humans, a strategy often called GOFAI ("Good Old Fashioned AI", [23, 111]). The computational theory of mind argued that if intelligence was reasoning operations in the mind, computers performing similar computations could also be intelligent [130, 148]. Purely symbolic artificial intelligence were often disconnected from the physical world, requiring symbolic representations as input, creating problems with grounding symbols in perception [75, 159] and often leading to brittleness [110, 104, 52]. However, symbolic reasoning problems themselves often proved to be relatively easy, whereas the physical problems of perceiving the environment or acting in it were actually the most challenging: what is unconscious for humans often requires surprising intelligence, often known as Moravec’s Paradox [68, 2]. Some researchers challenged this approach, arguing that for machines to be intelligent, they must interact with noisy environments via rich sets of sensors and effectors, creating high-bandwidth interactions that break the assumption of clean inputs and outputs and discrete states required by *classical AI* [181]; these ideas were echoed by roboticists already concerned with connecting sensors and actuators more directly [10, 25, 118]. Much as neural network concepts hibernated through several AI winters before enjoying a renaissance, embodied AI ideas have now been revived by new interest from fields such as computer vision, machine learning and robotics - often in combination with neural network ideas. New generations of artificial neural networks are now able to digest raw sensor signals, generate commands to actuators, and autonomously learn problem representations, linking "classical AI" tasks to embodied setups.

Thus, embodied AI is more than just the study of agents that are active and situated in their environments: it is an exploration of the properties

of intelligence. Embodied AI research has demonstrated that intelligent systems that perform well at embodied tasks often look different than their passive counterparts [64] - but, conversely, that highly performing passive AI tasks can often contribute greatly to embodied systems as components [156]. Furthermore, the control over embodied agents provided by modern simulators and deep learning libraries enables ablation studies that reveal fine-grained details about the properties needed for individual embodied tasks [123].

What is *not* Embodied AI? Embodied AI overlaps with many other fields, including robotics, computer vision, machine learning, and simulation. However, differences in focus make embodied AI a research area in its own right.

All *robotic* systems are embodied; however, not all embodied systems are robots (e.g., AR glasses), and robotics requires a great deal of work beyond purely trying to make systems intelligent. Embodied AI also includes work that explores the properties of intelligence in realistic environments while abstracting some of the details of low-level control. For example, the ALFRED [155] benchmark uses simulation to abstract away low-level robotic manipulation (e.g. moving a gripper to grasp an object) to focus on high-level task planning. Here, the agent is tasked with completing a natural language instruction, such as *rinse the egg to put it in the microwave*, and it can open or pickup an object by issuing a high-level *Open* or *Pickup* action that succeeds if the agent is looking at the object and is sufficiently close to it. Additionally, [123] provides an example of studying properties of intelligence, where they attempt to answer whether mapping is strictly required for a form of robotic navigation. Conversely, robotics includes work that focuses directly on the aspects of the real world, such as low-level control, real-time response, or sensor processing.

Computer vision has contributed greatly to embodied AI research; however, computer vision is a vast field, much of which is focused purely on improving performance on passive AI tasks such as classification, segmentation, and image transformation. Conversely, embodied AI research often explores problems that require other modalities with or without vision, such as navigation with sound [41] or pure LiDAR images.

Machine learning is one of the most commonly used techniques for building embodied agents. However, machine learning is a vast field encompassing primarily passive tasks, and most embodied AI tasks are formulated in such a way that they are learning agnostic. For example, the iGibson 2020 challenge [153] allowed training in simulated environments but deployment in holdout

environments in both real and simulation; nothing required the solutions to use a learned approach as opposed to a classical navigation stack (though learned approaches were the ones deployed).

Artificial intelligence is written into the name of embodied AI, but the field of embodied AI was created to address the perceived limitations of classical artificial intelligence [129], and much of artificial intelligence is focused on problems like causal reasoning or automated programming which are hard enough without introducing the messiness of real embodiments. More recently, techniques from more traditional artificial intelligence domains like natural language understanding have been applied to embodied problems with great success [3].

Simulation and embodied AI are intimately intertwined; while simulations of real-world systems go far beyond the topics of robotics, and the first generation of embodied AI focused on robotic embodiments [129], much of modern embodied AI research has expanded to simulated benchmarks, emulating or even scanned from real environments, which provide challenging problems for traditional AI approaches, with or without physical embodiments. Despite not starting with robots, systems that have resulted from this work have nevertheless found success in real-world environments [197, 164], providing hope that simulated benchmarks will prove a fruitful way to develop more capable real-world intelligent systems.

Why focus on real-world environments? Many researchers are exploring intelligence in areas such as image recognition or natural language understanding where at first blush interaction with an environment appears not to be required. Genuine discoveries about intelligent systems appear to have been made here, such as the role of convolutions in image processing and the role of recurrent networks and attention in language processing. So a reasonable question is, why do we need to focus on interactive and realistic (if not real-world) environments if we want to understand intelligence?

Focusing on interactive environments is important because each new modality of intelligence we consider - classification, image processing, natural language understanding, and so on - has required new architectures for learning systems [69], [48]. Interacting with an environment over time requires the techniques of reinforcement learning. Deep reinforcement learning has made massive strides in creating learning systems for synthetic environments, including traditional board games, Atari games, and even environments with simulated physics, such as the Mujoco environments.

However, embodied AI research focuses on environments that are either more realistic [188] or which require actual deployments in the real world [1,

153]). This shift in emphasis has two primary reasons. First, many embodied AI researchers believe that the challenges of realistic environments are critical for developing systems that can be deployed in the real world. Second, many embodied AI researchers believe that there are genuine discoveries to be made about the properties of intelligence needed to handle real-world environments that can only be made by attempting to solve problems in environments that are as close to the real world as is feasible at this time.

1.2 Common-Sense Knowledge for EAI.

When dealing with agents operating in indoor environments, common-sense knowledge has a central role. Common-sense knowledge is a type of knowledge that is considered essential for understanding and navigating the everyday world. It includes things like the basic properties of objects, the causes and effects of events, and the typical ways in which people and animals behave. When humans move in unknown environments, they largely exploit, for example, semantic information about object displacement learned from the experience. Usually, a bed is located in the bedroom, the toilet is in the bathroom, and the stoves are in the kitchen. In known environments, instead, humans can efficiently reuse previously acquired knowledge. For example, we all remember the layout of our best friend's house, and we don't need to ask every time where is the kitchen when we are thirsty. In crowded environments like offices, humans need to comply with some commonsense social rules, like avoiding collisions with others when moving or waiting outside a room and letting the other go out before entering. These examples, taken from innate human intelligence, immediately pose three research questions explored in this thesis:

- **R1:** *How can an agent exploit common-sense knowledge about the environments?*
- **R2:** *How can an agent reuse previously acquired knowledge about a specific environment?*
- **R3:** *How can an agent comply with social rules?*
- **R4:** *How can an agent acquire knowledge and common-sense rules?*

For **R1**, it is necessary to tackle two sub-questions: (i) Which common-sense knowledge is more useful for a particular task? (ii) How can an agent to exploit the knowledge? The choice of common-sense knowledge for an embodied AI agent is strongly related to the task that the agent will tackle. For example,

knowledge about the spatial relationships and co-locations of objects and rooms may be useful for an agent that is tasked with navigating to a specific location. However, this knowledge may not be as useful for an agent that is following natural language instructions. There is also a wide range of common-sense knowledge that can be useful in many scenarios, regardless of the specific task that the agent is tackling. For example, knowledge about physical dynamics and motion can be useful for an agent navigating a cluttered environment and avoiding collisions and obstacles. Concerning (ii), there are multiple possibilities for using common-sense knowledge effectively in an embodied AI agent. One option is to use a shared embedding between rooms and objects or to add an extra term to the reward function that encourages or penalizes the agent based on its use of common-sense knowledge. Another option is to use a static data structure, such as a matrix, to encode the common-sense knowledge in a modular model. Overall, the choice and use of common-sense knowledge in an embodied AI agent is a complex challenge and requires careful consideration of the specific task, environment, and context in which the agent operates.

Addressing **R2**, i.e., Reusing previously acquired knowledge in a specific environment, presents multiple challenges. For example, (i) methods for representing and storing knowledge can be exploited in a structured and accessible way so that the agent can easily retrieve and use the knowledge when needed. This could involve using knowledge representation languages or ontologies to organize the knowledge and using indexing or other techniques to enable efficient access. Furthermore, the knowledge should be extensible. (ii) explore ways to integrate previously acquired knowledge with new observations and experiences so that the agent can update and refine its knowledge over time. This could involve using techniques such as Bayesian inference or probabilistic reasoning to combine prior knowledge with new evidence and update the knowledge considering the environment’s uncertainty and variability. Another potential study area is transferring knowledge from one environment to another so the agent can reuse knowledge from similar environments to improve performance in a new environment. This could involve studying the transferability of knowledge across different domains and developing methods for adapting the knowledge to the new environment’s specific characteristics. Another option is to examine the role of memory and attention in enabling the agent to reuse its environmental knowledge. This could involve studying how the agent’s memory and attention mechanisms can selectively focus on relevant information, retain and retrieve important knowledge, and optimize these mechanisms to support efficient knowledge reuse. Instead, in long-horizon problems, where, for example, the agent has to search a large number of objects in an environment in a given order also

simpler representations can be exploited. In 4 we will show an agent that exploits a very simple memory to store and reuse information acquired during the environment exploration.

To force an embodied AI agent to respect common-sense social rules auxiliary (i.e., extra heads in a model that are used to refine some features) tasks that directly model social-rules can be exploited. These tasks require a combination of sensory perception, data analysis, and decision-making algorithms. The agent can use sensors, such as cameras and microphones, to perceive its surroundings and to detect other agents, objects, and obstacles in the environment. It can then use data analysis algorithms, such as computer vision and natural language processing, to interpret the sensory data and infer other agents' intentions, goals, and behaviors. Based on this information, the agent can use decision-making algorithms, such as planning and reinforcement learning, to plan its own actions and movements in a way that complies with the common-sense social rules of the environment. For example, the agent could use social rules, such as "don't bump into other agents" or "don't block the path of other agents," to guide its actions and avoid conflicts and violations of social norms. In addition, the agent can use auxiliary losses to regularize its decision-making process and to encourage it to learn the common-sense social rules of the environment. For example, the agent could be penalized with an auxiliary loss whenever it violates a social rule, such as bumping into another agent or blocking its path. This can provide a strong incentive for the agent to adopt the social rules and avoid violating them in the future. Overall, using auxiliary losses to force an embodied AI agent to respect common-sense social rules requires a combination of sensory perception, data analysis, and decision-making algorithms, which can be tailored to the specific needs and requirements of the agent and its environment. In Chapter 5[29], we will explore a work where we exploited two different auxiliary tasks that model the *collision danger* between the agent and humans.

Finally, concerning **R4** not always is common-sense knowledge available and ready to use in a model. For example, it can be difficult to directly translate human knowledge into symbolic rules or quantified probabilities. In such scenarios, it would be useful to learn explicit rules in an end-to-end fashion. Let's say there is a problem where an agent needs to find a specific object in an environment and only the labels of the rooms are available. One option is to use a pre-trained model to detect the objects, but this can lead to issues with domain transfer or the general transfer capabilities of the model. Another option is to use weak supervision on the room type to understand and learn that the presence of certain objects (e.g. a bed or a cabinet) increases the probability of being in a specific room (e.g., $cabinet(x) \wedge bed(x) \implies bedroom(x)$). This falls within the domain of Neural

Symbolic Integration (NeSy), where a neural network is used to produce symbolic output that the model can then use for symbolic reasoning.

In summary, this introduction has discussed the central role of common-sense knowledge in agents operating in indoor environments. This knowledge allows humans to navigate effectively in unknown environments, exploiting semantic information about object displacement. In known environments, humans must efficiently reuse previously acquired knowledge. However, in crowded environments like offices, agents need to follow social rules like avoiding collisions and waiting outside rooms. These examples raise three research questions: how can an agent exploit common-sense knowledge about the environment, how can it reuse previously acquired knowledge, and how can it comply with social rules in cluttered environments. Finding answers to these questions is a complex challenge that requires careful consideration of the specific task, environment, and context in which the agent operates. In the following chapters will be explored directions to solve these questions.

1.3 Contributions

In the following chapters we discuss all the contributions done in the different works investigated and here we reported a summary of each contribution:

Exploiting scene-specific features for object goal navigation (R1).

Can the intrinsic relation between an object and the room in which it is usually located help agents in the Visual Navigation Task? We study this question in the context of Object Navigation, a problem in which an agent has to reach an object of a specific class while moving in a complex domestic environment. In Chapter 2, we will introduce a new reduced dataset that speeds up the training of navigation models, a notoriously complex task. Our proposed dataset permits the training of models that do not exploit online-built maps in reasonable times even without the use of huge computational resources. Subsequently, we propose a model capable of exploiting the subtle relationship between the classification of the scenes and the objects contained in them, highlighting quantitatively that the idea is correct.

Online Learning of Reusable Abstract Models for Object Goal Navigation (R2).

In Chapter 3, we present a novel approach to incrementally learn an Abstract Model of an unknown environment, and show how an agent can reuse the learned model for tackling the Object Goal Navigation task. The Abstract Model is a finite state machine in which each state is an abstraction of a state of the environment, as perceived by the agent in a

certain position and orientation. The perceptions are high-dimensional sensory data (e.g., RGB-D images), and the abstraction is reached by exploiting image segmentation and the Taskonomy model bank. The learning of the Abstract Model is accomplished by executing actions, observing the reached state, and updating the Abstract Model with the acquired information. The learned models are memorized by the agent, and they are reused whenever it recognizes to be in an environment that corresponds to the stored model. We investigate the effectiveness of the proposed approach for the Object Goal Navigation task, relying on public benchmarks. Our results show that the reuse of learned Abstract Models can boost performance on Object Goal Navigation.

Modular-MON: Modular Multi-Object Navigation (R2). Chapter 4 focuses on the Multi-Object Navigation (MultiON) task, where an agent needs to navigate to multiple objects in a given sequence. We systematically investigate the inherent modularity of this task by dividing our approach into four modules: (a) an *object detection* module trained to identify objects from RGB images, (b) a *map building* module to build a semantic map of the observed objects, (c) an *exploration* module enabling the agent to explore its surroundings, and finally (d) a *navigation* module to move to identified target objects. We focus on the navigation and the exploration modules in this work. We show that we can effectively leverage a PointGoal navigation model in the MultiON task instead of learning to navigate from scratch. Our experiments show that a PointGoal agent-based navigation module outperforms analytical path planning on the MultiON task. We also compare exploration strategies and surprisingly find that a random exploration strategy significantly outperforms more advanced exploration methods.

Exploiting Socially-Aware Tasks for Embodied Social Navigation (R1, R3). Learning how to navigate among humans in an occluded and spatially constrained indoor environment, is a key ability required to embodied agent to be integrated into our society. In Chapter 5, we propose an end-to-end architecture that exploits Socially-Aware Tasks (referred as to Risk and Social Compass) to inject into a reinforcement learning navigation policy the ability to infer common-sense social behaviors. To this end, our tasks exploit the notion of immediate and future dangers of collision. Furthermore, we propose an evaluation protocol specifically designed for the Social Navigation Task in simulated environments. This is done to capture fine-grained features and characteristics of the policy by analyzing the minimal unit of human-robot spatial interaction, called Encounter. We validate our approach on Gibson4+

and Habitat-Matterport3D datasets.

Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions (R4). Neuro-Symbolic (NeSy) integration combines symbolic reasoning with Neural Networks (NNs) for tasks requiring perception and reasoning. Most NeSy systems rely on continuous relaxation of logical knowledge, and no discrete decisions are made within the model pipeline. Furthermore, these methods assume that the symbolic rules are given. In Chapter 6, we propose *Deep Symbolic Learning* (DSL), a NeSy system that learns *NeSy-functions*, i.e. the composition of a (set of) *perception functions* which map continuous data to discrete symbols, and a *symbolic function* over the set of symbols. DSL learns simultaneously the perception and symbolic functions, while being trained only on their composition (NeSy-function). The key novelty of DSL is that it can create internal (interpretable) symbolic representations and map them to perception inputs within a differentiable NN learning pipeline. The created symbols are automatically selected to generate symbolic functions that best explain the data. We provide experimental analysis to substantiate the efficacy of DSL in simultaneously learning perception and symbolic functions.

Chapter 2

Exploiting Scene-Specific Features for Object Goal Navigation

In this chapter, we aim to address **R1** (How can an agent exploit common-sense knowledge about the environments?). To this end, we present a model [27] for the Object Goal Navigation task, in which an agent must locate a specific object within an unknown environment.

To accomplish this task, our model creates a shared embedding between the goal category (e.g., a bed, a sink) and the features extracted from an RGB image by a semantic classifier trained to classify the room type (e.g., a kitchen, a bedroom). By learning this shared representation, the agent can indirectly learn that certain objects are typically located in certain rooms. This shared representation is then used to inform the agent's navigation decisions along with a Scene-Memory Transformer model [62]. We found that the learned common-sense knowledge increases the agent's efficiency in locating the desired object.

2.1 Introduction

¹Visual Navigation is a trending topic in the Computer Vision research community. This growth in interest is undoubtedly due to the important practical implications that the development of agents capable of moving in complex environments can have on our society. For example, shortly, we will be able to ask robotic assistants to perform the most disparate tasks in our homes. Before we can ask a robot to take something out of the refrigerator,

¹**T. Campari**, P. Eccher, L. Serafini, L. Ballan. "Exploiting scene-specific features for object goal navigation", *European Conference on Computer Vision, Glasgow, 2020, (ECCVW 2020)*

however, we need to make sure that it can find the fridge and get to it while avoiding the complex tangle of obstacles that a domestic environment can contain. For this reason, this work focuses on the Object Navigation task, defined in [6] as the search for objects belonging to a specific class by a robotic agent. For humans, this is a straightforward task, whatever the object to be found is. A human can build a mental link between the object and the room where it is more likely to be found. In this way, a human can simplify the problem by first searching for the room and then for the required object inside the room. For example, think of having to look for a sink; unconsciously, we will first search for the kitchen or bathroom, and then we will find a sink in them. To implement this intuition, we have developed an attention-based [170] policy in which we exploit a joint representation that integrates inside it visual information extracted with a scene classifier and encoding of the semantic goal to be searched. This representation allows us to increase performance compared to other models taken into consideration significantly.

Several works tried to tackle Navigation through Learning models [36, 33]. In particular, [36] leveraged depth images to construct, in an online fashion, semantic maps of the environment. From these maps, they tried to maximize the exploration of the scene. To do this, they placed intermediate subgoals in unexplored areas of the map that the agent was encouraged to reach through planning. This approach inevitably tends to lengthen the agent's paths, at least until the object sought is visible, since wanting to maximize exploration involves a significant amount of moves by the agent. For the simpler PointGoal Navigation [6] task, proposed in the "Habitat Challenge 2019"², it was possible to observe how a simple model [180] based on LSTM was able to perform better than competitors based on more complex architectures that exploit maps creation [73, 35]. This was made possible by the DD-PPO algorithm [180], a distributed version of the PPO [147] Reinforcement Learning algorithm capable of parallelizing learning massively. In fact, they used 64 NVIDIA V100 GPUs for three consecutive days of training. In other words, the model was trained for about 180 days in a single GPU setting. However, not all researchers can access those massive hardware resources. For this reason, we have generated a reduced version of the dataset produced for the "Habitat Challenge 2020"³ for the Object Navigation task that would allow the training of Deep Reinforcement Learning models in a few hours. In this way, even using few computational resources, complex models can be trained, which on the original dataset would take days.

²<https://aihabitat.org/challenge/2019/>

³<https://aihabitat.org/challenge/2020/>

Furthermore, in [62], it was pointed out that Recurrent Neural Networks are not recommended in Navigation tasks. These structures usually have the issue of considering as more critical the recent past and, at the same time, gradually forgetting the remote past. On the contrary, by exploiting the principle of attention, described in [170], it is possible to record all the past observations in a memory from which to extract information on every single step that the agent has undertaken in the past, improving that highly penalizing intrinsic aspect in the behavior of the Recurrent Neural Networks.

Summarizing, our contribution is twofold:

- We propose a new reduced dataset for the Object Navigation task extracted from the one proposed in the "Habitat Challenge 2020", on which it is possible to test algorithms that would require a lot of resources for training, and that maintain as far as possible the main characteristics that the first had;
- An attention-based policy for Object Navigation that can exploit, starting from RGB images only, the intuition mentioned above, namely that a correlation exists that binds objects to specific rooms.

2.2 Related Works

Visual Navigation is an increasingly central topic within Computer Vision. However, Visual Navigation involves several different sub-problems. In this section, we will summarize the most relevant related works to *Simulators and 3D Datasets for Visual Navigation* and other different research areas connected to *Visual Navigation*.

Simulators and 3D Datasets for Visual Navigation In recent years a large number of different simulators have been developed. GibsonEnv [185, 186] and AI2Thor [90] both allow to simulate multi-agent situations and to interact with objects, for example, lifting them, pushing them, etc. Matterport3DSimulator [7] can provide the agent with photorealistic images extracted from Matterport3d [32] and is mainly used for Room2Room Navigation problem. HabitatAI [145], instead, provides support to 3D datasets such as Gibson, Matterport3D, and Replica [160].

Visual Navigation Also, thanks to the new possibilities offered by these simulators today, numerous tasks are available, as pointed out in [6]. Common Navigation tasks are mainly divided into two categories, namely those that

require active exploration of the environment and those that, on the other hand, provide tools that can signal, for example, via GPS sensors, the direction to be taken to reach the required goal.

In Classical Navigation, numerous approaches perform path planning on explicit maps [86, 30, 99]. More recently, however, approaches based on Reinforcement Learning have been presented through policies based on Recurrent Neural Networks [113, 98, 144, 62, 120]. Mirowski et al. [113] define an approach that jointly learns the goal-driven Reinforcement Learning problem with auxiliary depth prediction and loop closure classification tasks by exploiting the A3C algorithm [116]. Mousavian et al. [120] propose a Deep Reinforcement Learning framework that uses an LSTM-based policy for Semantic Target Driven Navigation. But LSTMs when they have to analyze very long data sequences, tend to focus more on the most recent observations, giving less importance to the first ones that have been seen. On the contrary, Fang et al. [62] propose Scene Memory Transformer, a policy based on attention [170] that can exploit even the least recent steps performed by the agent. In this case, the policy training is conducted with the Deep Q-Learning algorithm [115]. Starting from the work done in [198], Sax et al. [146] show that using Mid-Level Vision results in policies that learn faster and generalize better when compared to learning from scratch. The Mid-Level model achieved high results in the PointGoal Navigation task. In [180], a scalable Reinforcement Learning algorithm on multiple GPUs capable of solving the PointGoal Navigation task almost perfectly has been presented. This solution, in particular, shows how Visual Navigation is a complex task that requires an impressive amount of resources. Their training was conducted on 64 GPUs for three days. Unfortunately, these resources are not within reach of the scientific community, and training similar models remain almost prohibitive for most researchers.

Some works have recently been presented for the Target Driven Navigation, such as [183, 192, 184]. Wu et al. [184] construct a probabilistic graphical model over the semantic information to explore structural similarities between the environment. Yang et al. [192], instead, propose a Deep Reinforcement Learning model that exploits the relationships between objects, encoded through a Graph Convolutional Network, to incorporate semantic priors within the policy. Chaplot et al. [36] propose a model for the ObjectGoal Navigation that constructs, during the exploration, a map with the semantic information of the scene extracted through a semantic segmentation model; from the generated map, a long-term goal is selected to maximize exploration, through a policy trained with Reinforcement Learning, when the searched object is visible it is set as a new long-term goal. The agent's actions are selected through the use of the fast marching algorithm [151].

2.3 Dataset

Previous works [180] have been able to achieve excellent results on the Point Navigation task [6], a simpler Navigation problem that doesn't require Semantic capabilities while using an architecture that didn't include complex components such as occupancy maps [43]. However, they leveraged massive parallelism using hardware resources inaccessible to most institutions. To cope with our shortage of dedicated GPU resources - we had access to a single GPU for our experiments - we decided to concentrate on a subset of the Matterport3D Dataset [32]. We argue that our choice of such a subset still offers significant results as its statistical indicators are similar to the original Matterport3D.

The extraction of the subset was done by restricting the problem to 5 out of 21 objects, choosing **Chair**, **Cushion**, **Table**, **Cabinet**, **Sink**. These objects are among the most frequent in the original set, as is shown in Figure 2.1. We decided to include the **Sink** object as it is a characterizing element of the bathroom. This increase the diversity of the domestic environments represented by our proposed dataset.

Related to the distributions of objects, one of the main issues in evaluating Object Navigation agents using Matterport3D is the Long Tail Distribution that limits the number of instances of infrequent objects seen during training. This, combined with metrics that ignore the precision of single classes, may undermine the development of agents with truly semantic capabilities. Furthermore, we decided to extract our new dataset - from here; we will refer to it as *Small MP3D* - using 6 out of 56 scenes of the official Training split. These scenes are `r47D5H71a5s`, `i5noydFURQK`, `ZMojNkEp431`, `jh4fc5c5qoQ`, `HxpKQynjfin` and `GdvgFV5R1Z5`. For each object in each scene, we extracted 100 episodes for the training split and 20 for both validation and test sets. In total, Small MP3D possesses 3000 training episodes and 600 episodes for both validation and test. To ensure the ability to generalize to unseen scenes, we decided to generate also an Unseen Test and Validation Set, extracted from the `D7N2EKCX4Sj` and `aayBHfsNo7d` scenes, each with 100 episodes (10 episodes for each object class in each scene).

The characteristics of the new dataset are shown in Table 2.1. We report the average values of Euclidean and Geodesic Distance as well as the number of Steps required to complete the episodes. Overall, the complexity of the training split of the proposed reduced dataset is lesser than the original data as both the Geodesic distance and the required Number of Steps are lower. However, the complexity of the Unseen Test split is significantly higher, with 30% more required steps on average. We think that this additional complexity

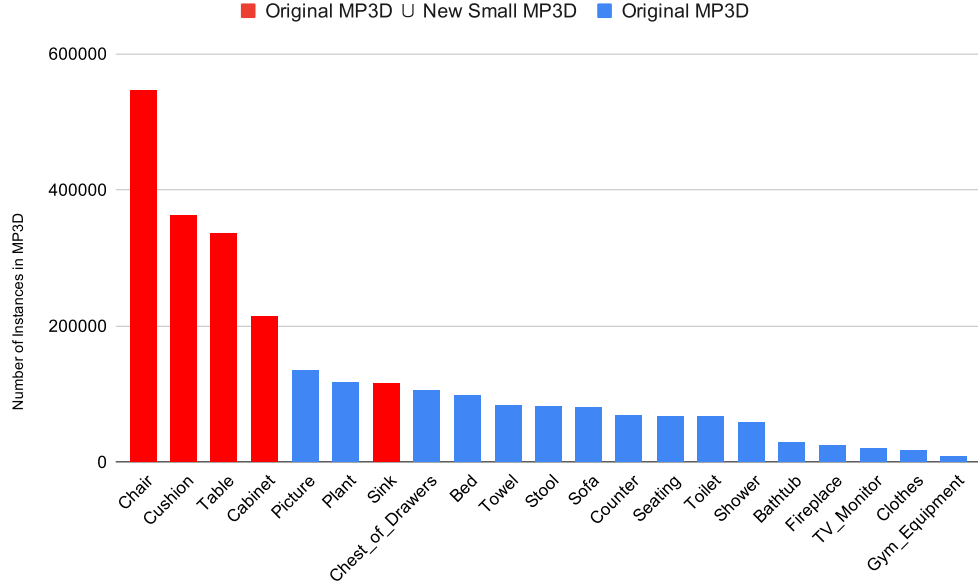


Figure 2.1: Distribution of objects in the Matterport3D Dataset.

	Chair			Cushion			Table		
	Euc	Geo	Steps	Euc	Geo	Steps	Euc	Geo	Steps
Original Train	5.17	6.45	40.05	5.39	7.23	45.56	4.98	6.19	38.82
New Train	3.50	4.00	27.71	5.79	7.23	49.14	3.12	3.85	28.42
Original Val	5.09	7.21	43.32	4.50	6.18	38.77	3.66	5.29	33.83
New Seen Val	3.43	3.89	27.01	6.09	7.58	51.37	2.77	3.42	26.42
New Unseen Val	4.07	5.79	37.15	9.00	12.19	68.40	4.53	5.18	32.95
New Seen Test	3.64	4.15	28.76	6.25	7.61	50.17	3.15	3.79	27.82
New Unseen Test	4.06	5.11	32.75	10.13	12.94	70.50	4.55	5.37	34.45
	Cabinet			Sink			Total Average		
	Euc	Geo	Steps	Euc	Geo	Steps	Euc	Geo	Steps
Original Train	5.24	6.84	42.63	6.33	8.54	51.80	5.27	6.78	42.27
New Train	4.02	4.91	34.15	4.94	5.99	39.80	4.27	5.19	35.84
Original Val	5.40	7.46	46.10	6.52	8.83	53.50	4.73	6.65	40.98
New Seen Val	4.02	4.87	34.07	4.62	5.69	38.22	4.19	5.09	35.42
New Unseen Val	5.80	6.76	46.65	7.17	8.87	52.25	6.11	7.76	47.48
New Seen Test	4.37	5.33	36.58	4.96	6.06	40.16	4.47	5.39	36.70
New Unseen Test	8.40	9.58	56.85	8.87	10.93	63.60	7.20	8.79	51.63

Table 2.1: Statistics of our dataset

can guarantee a fair and meaningful benchmark.

2.4 Method

In this section, we first describe the Problem Setup. Then we introduce our SMTSC model as shown in Fig. 2.2

2.4.1 Problem Setup

Our interests fall within the task of Object Navigation. In particular, this task requires finding an occurrence of an object starting from a random position in the environment.

The set of possible actions is defined as $\mathcal{A} = \{ \text{go_forward}, \text{turn_left}, \text{turn_right}, \text{stop} \}$. The actions are deterministic, that is, apart from possible collisions with objects in the scene, the agent will move in the desired direction without deviations due to noisy dynamics. The observation $o = (\text{RGB}, p, a_{\text{prev}}, \text{goal}) \in \mathcal{O}$ is the set of features collected by the agent at each step in the environment and passed to the model. RGB is what the agent sees from a given position, it is an RGB image extracted with 640x480 size; p is the agent’s position w.r.t. to the starting point, a_{prev} is the action performed in the previous step and finally goal is the objective object to be sought.

2.4.2 Model

The proposed model is visible in Figure 2.2, it is composed of two main parts, a first part in which the features are extracted and brought into a joint representation and a second module in which the features of the current observation are added to a memory that keeps track of all past observations and an attention-based policy network extracts a distribution on possible actions.

Features Processing Module Starting from an observation $o = (\text{RGB}, p, a_{\text{prev}}, \text{goal}) \in \mathcal{O}$ we first define 5 different encoders:

1. $\gamma_{\text{sem_seg}}(\text{RGB}) : \mathbb{R}^{256 \times 256} \rightarrow \mathbb{R}^{256}$: encodes RGB observations into a vector of size 256 by using features extracted from a semantic segmentation model.
2. $\gamma_{\text{scene_class}}(\text{RGB}) : \mathbb{R}^{256 \times 256} \rightarrow \mathbb{R}^{128}$: encodes RGB observations into a vector of size 128 by using features extracted from a scene classification model.
3. $\gamma_{\text{goal}}(\text{goal})$: encodes the goal into a vector of size 32

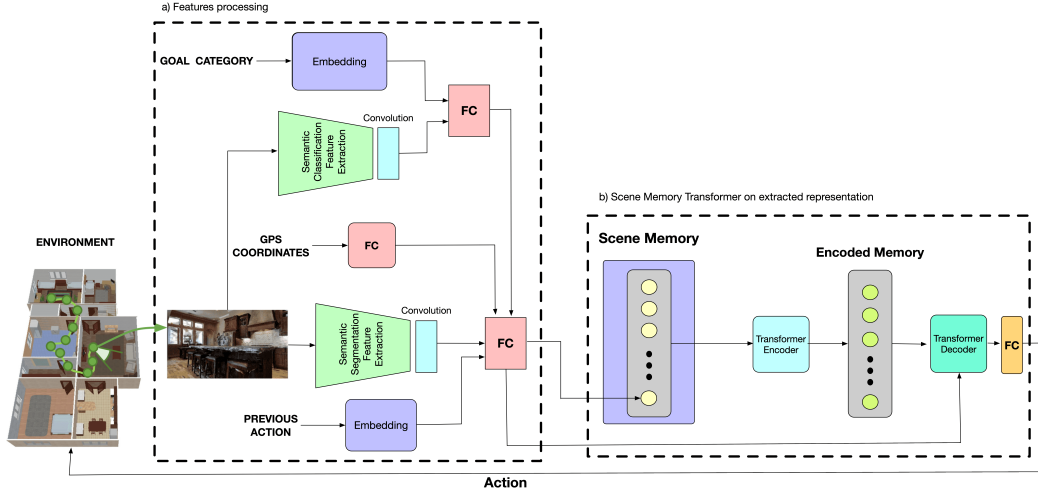


Figure 2.2: SMTSC model: a) Features processing: all features are processed and a shared representation is created. This representation is then inserted into a memory (b) Scene Memory Transformer) from which the action that the agent will perform is extracted with a Scene Memory Transformer model.

4. $\gamma_{\text{pos}}(p)$: encodes the relative position into a vector of size 32
5. $\gamma_{\text{act}}(a_{\text{prev}})$: encodes the previous action executed into a vector of size 32

Now we define:

$$\delta(RGB, goal) = FC(\{\gamma_{\text{scene_class}}(RGB), \gamma_{\text{goal}}(goal)\}) \quad (2.1)$$

And:

$$\phi(o) = FC(\{\gamma_{\text{sem_seg}}(RGB), \gamma_{\text{pos}}(p), \gamma_{\text{act}}(a_{\text{prev}}), \delta(RGB, goal)\}) \quad (2.2)$$

Eq. 2.1 encodes the previously illustrated idea that a goal is intrinsically associated with a specific room. To do this, starting from the goal through the γ_{goal} function, a representation of dimension 32 is extracted. In parallel, a scene classifier is used to extract features starting from the RGB image, these two modalities are concatenated and a joint representation is created using a fully connected layer. In this way, we can obtain a representation of the goal conditioned step by step from the room in which the agent is located, which is going to add helpful information to the agent to understand how to move in the environment.

Finally, Eq. 2.2 generates a joint representation between all the modalities described above. It, therefore, concatenates the representation obtained from a model for semantic segmentation and the representations of the previous

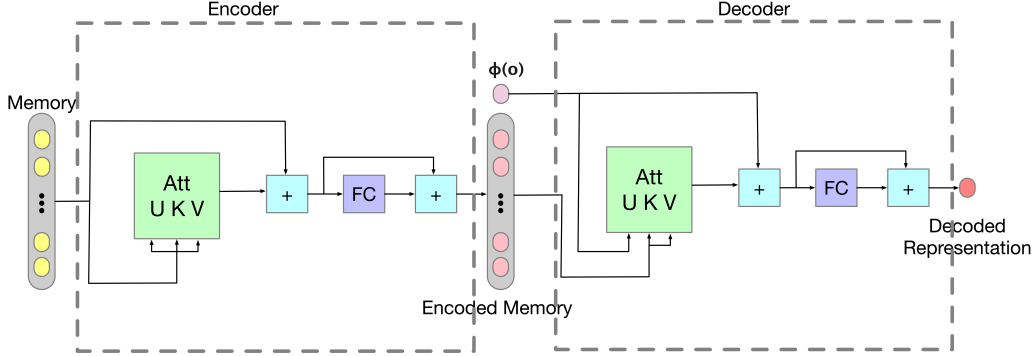


Figure 2.3: The structure of the encoder-decoder model based on Multi-Head Attention, as presented in Fang et al.[62].

action, position, and goal (as defined by Eq. 2.1). This vector is passed to a fully connected layer which returns a vector of size 256.

Scene Memory Transformer Module This module is based on the one proposed by [62]. Given a new episode, we build an initially empty memory where to save the joint representations obtained starting from observation $o = (\text{RGB}, p, a_{\text{prev}}, \text{goal})$ through the application of the $\phi(o)$ function defined in the Eq. 2.2. This memory is then passed along with $\phi(o)$ to an attention-based encoder-decoder policy [170] which extracts a probability distribution on the actions. The encoder-decoder structure is shown in Figure 2.3. The SMT encoder uses a self-attention to encode the M memory, so M is passed to a MultiHeadAttention with eight heads; therefore, M is both Query and Key and Value as shown in Eq. 2.3.

$$M' = \text{MultiHeadAttention}(M, M) \quad (2.3)$$

Subsequently, M is passed with the joint representation to the decoding structure. It always uses the attention mechanism to give a representation of $\phi(o)$ conditioned by past observations. Again, attention is implemented through an 8-head MultiHeadAttention mechanism as shown in Eq. 2.4.

$$Q = \text{MultiHeadAttention}(\phi(o), M') \quad (2.4)$$

Finally, Q is reduced to the dimensionality of the action space \mathcal{A} through a Linear Layer and a Categorical Distribution on the action space is extracted from the latter representation, as defined in Eq 2.5.

$$\pi(a|o, M) = \text{Categorical}(\text{Softmax}(\text{FC}(Q))) \quad (2.5)$$

Implementation and Training Details We implemented all the models using Python 3.6, and PyTorch [124]. We used the PPO [147] algorithm to train the model using a 32 GB Tesla V100 GPU. We used a batch size of 64 and Adam Optimizer with a Learning Rate of 1×10^{-5} . The visual features are extracted from the images using the Taskonomy networks [198] as done in [146]. This allows us to have consistent features as Taskonomy networks have been trained on indoor environments simulated by Gibson [185] and also, the number of parameters to be trained on the network drops drastically. All the activation functions within the encoder-decoder structure are ReLU functions, and the pose vector is encoded as a quadruple $(x, y, \sin(\theta), \cos(\theta))$.

2.5 Experimental Results

In this section an accurate description of the experimental setup will be provided, the results obtained will then be presented.

2.5.1 Experimental setup

We used the *Small MP3D* dataset presented in Section 2.3. We decided to test the SMTSC model on both the seen and unseen test sets. Results on the seen set will give as a measure of its capacity to memorize environments seen during training. Conversely, results obtained on the unseen set will quantify the degree of adaptation to unseen scenes that the agent possess. However, as the number of training scenes is only six, we don't expect our agents to develop strong generalization behaviours. The simulator used for all the experiments was Habitat, which provides the agent with 640x480 RGB images. In addition to the images, an odometry system is available that can provide the x and y coordinates and the orientation of the agent with respect to the starting point. The simulated robot has a height of 88 cm from the ground with a radius of 18cm. The camera of the agent with which he acquires the images is placed 88cm from the ground and allows to capture images with a 79° HFOV. Sliding against objects is not allowed, once a collision has been made the agent must necessarily rotate before being able to proceed again in the environment. The metrics used to evaluate the proposed models are 3: Success Rate, Success weighted by Path Length (SPL) and distance to success (DTS), as defined in section 3.4. Apart from the model presented in Section 2.4, four other different baselines have been tested to assess the performance of the proposed model.

Random Agent: an agent that performs random actions extracted from a uniform distribution.

Model	SPL \uparrow	Success \uparrow	DTS \downarrow
Random	0.00	0.00	5.126
Forward-Only	0.009	0.026	5.094
Reactive [146]	0.041	0.126	4.523
LSTM [180]	0.131	0.247	3.199
SMT w/o SC (our)	0.345	0.595	1.562
SMTSC (our)	0.649	0.883	0.403

Table 2.2: Results on the Seen Test Set

Model	SPL \uparrow	Success \uparrow	DTS \downarrow
Random	0.00	0.00	7.842
Forward-Only	0.004	0.01	7.922
Reactive [146]	0.001	0.04	7.797
LSTM [180]	0.002	0.04	7.648
SMT w/o SC (our)	0.008	0.04	7.518
SMTSC (our)	0.039	0.080	6.817

Table 2.3: Results on the Unseen Test Set

Forward-Only Agent: an agent who only performs forward actions (with a 1% probability of calling the stop action). These first two baselines were placed to demonstrate the non-triviality of the dataset proposed.

Reactive Agent: a policy that extracts semantic segmentation features through Taskonomy and merges them with position and goal as was done in [146]. The action to be performed is directly extracted from this representation, therefore no type of memory is used.

LSTM Agent: the representation is extracted as for the reactive agent, but in this case, it is passed to an LSTM and the action to be performed is extracted from its output. The model is pretty similar to the RGB-Only presented in [180].

SMT without Scene Classification Features (SMT w/o SC): this is the same model presented in Section 2.4, except for the fact that the goal is coded only through an embedding layer without exploiting the joint representation with the features of the Scene Classifier.

The last three models were trained with PPO Reinforcement Learning algorithm.

2.5.2 Results

In Table 2.2 are shown the results obtained on the test set in seen environments. The low performance of Random and Forward-Only baselines highlight how the dataset is not trivial, showing that the object to be found is almost never in the immediate proximity of the starting point of the episode. Looking instead at the other baselines based on Reinforcement Learning, we can see how the two models that use the Scene Memory Transformer are able to perform much better in almost all metrics. This big difference is probably attributable to the fact that the SMT can extract crucial information even from actions performed in a fairly remote past, while for example in the case of the LSTM this is very difficult, as more recent information tends to supplant the older ones and this is emphasized especially in very long sequences.

It is also interesting to note that the model that creates a joint representation between the goal coding and the visual features for the scene classification performs much better than the SMT w/o SC model. Even in the SPL, we have an increase of 88%, in the success ratio an increase of 48.4% and finally, the average DTS has fallen by over a meter.

The behavior observed in the seen test set follows the same trend also in the case of the unseen test set whose results are visible in Table 2.3. In fact, taking as a comparison the average geodesic distance of the unseen test set (7.76m), we can see how the Forward Only and Random agents tend to conclude their episodes even further away from the starting point. The reactive model instead certifies its performance in line with the average distance reported in Table 2.1 for the unseen test set. Finally, the two models that exploit the SMT are the ones that perform better, even in an unseen environment. In particular, the proposed model capable of exploiting the joint representation between visual features and goal coding lowers the average distance from the goal by almost a meter, and certifies its successes on 8% of the scenes.

In the next section we report some qualitative example of the SMTSC model on the seen and unseen test sets.

Qualitative Results In Figure 2.4 it is possible to see, in blue, the path taken by the agent to reach an object of the "cushion" class on the seen test set. In green you can see the shortest path to the object. The agent has successfully reached the object sought by stopping less than 0.1m from it. On the contrary, in Figure 2.5 is shown an example of failure, still on the seen test set, in which the agent was unable to reach the object sought. The agent, from his initial position was able to recognize an object of the class sought and to head towards it, despite not being the closest chair. At the time of calling the stop action, however, the agent was 7cm beyond the boundary



Figure 2.4: Successful navigation episode with SMTSC model on seen test set.

that would have given the episode success. In this case, we can see the strong penalty given by a metric like the SPL, in fact, in our opinion, this episode cannot be considered totally wrong.

In Figures 2.6 and 2.7, on the other hand, it is possible to see two examples extracted from the evaluation of the model on the unseen test set. In the first image, the agent was able to take a correct path to a "cabinet". However, the cabinet that was found wasn't the nearest so the SPL of this episode was only 0.57. In the second example, on the other hand, a very long route of over 14 meters of navigation is presented. The agent here was able to follow the optimal path for about half of its length, then it reached the maximum number of actions allowed. This means that it "wasted" a lot of action to increase its understanding of the scene.

2.6 Conclusion

We proposed a subset of the dataset developed for the Habitat 2020 ObjectNav Challenge [145]. This was done to allow the training of models that do not involve the use of planning associated with the construction of maps within them with few computational resources (e.g. a single GPU). Furthermore, we proposed a model capable of exploiting the subtle relationship existing between objects and the rooms in which they are usually located. This intuition, combined with the use of a Scene Memory Transformer showed good results on the proposed dataset. In the future, it would be very interesting

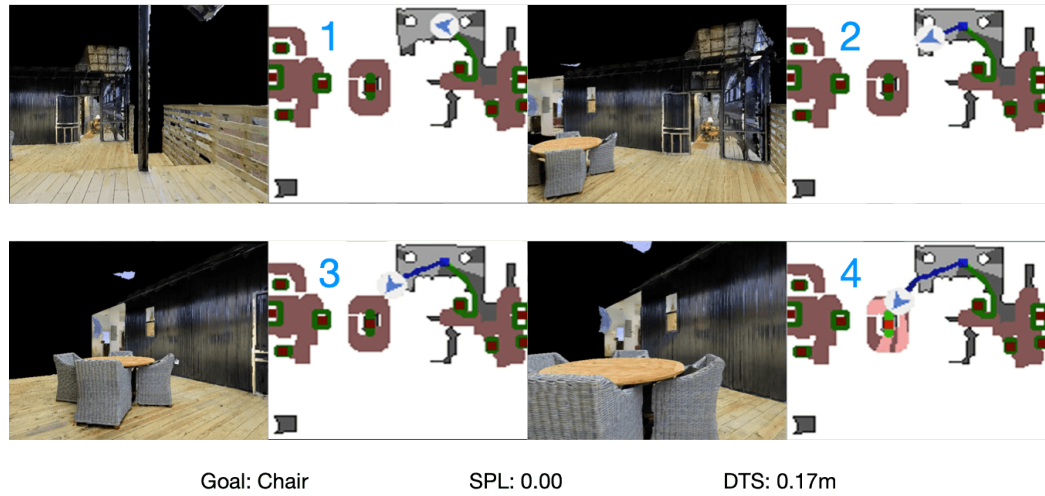


Figure 2.5: Unsuccessful navigation episode with SMTSC model on seen test set.

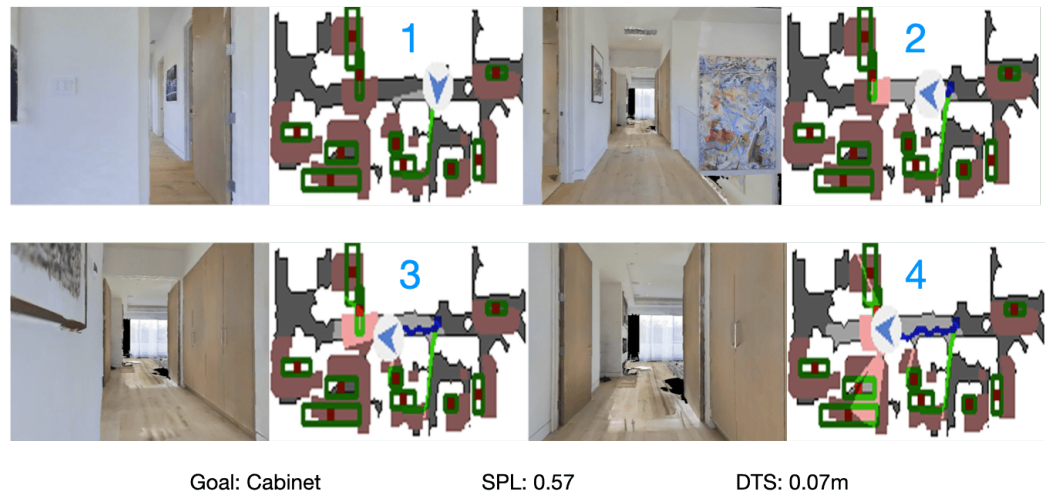


Figure 2.6: Successful navigation episode with SMTSC model on unseen test set.

to be able to test this model on the complete dataset using the distributed Reinforcement Learning algorithm DD-PPO [180] and a greater number of GPUs in order to perform training in a reasonable time.

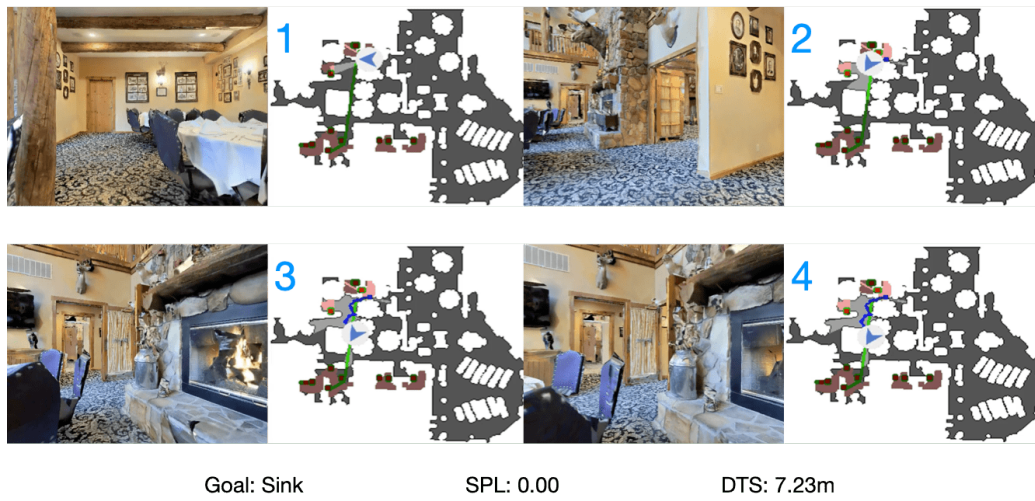


Figure 2.7: Unsuccessful navigation episode with SMTSC model on unseen test set.

Chapter 3

Online Learning of Reusable Abstract Models for Object Goal Navigation

In this chapter, we tackle a modified variant of the Object Goal Navigation task, where the agent can reuse previously acquired knowledge about a specific environment (**R2**). We propose a modular approach [28] for the Object Goal Navigation task, where an agent interleaves exploration and exploitation phases. In the exploration phase, the agent acquires new information, which are stored in an Abstract Model. In the exploitation phase the agent *exploits* the information recorded in the Abstract Model to efficiently reach the sought object goal. The agent is able to reload previously acquired knowledge relevant to the current environment whenever it recognizes to be in a previously explored environment. This is defined as the relocation problem.

3.1 Introduction

¹In Embodied AI, agent's intelligence emerges from the interaction with the environment as the result of sensorimotor activities [157]. While acting in a real environment, an agent should acquire and effectively represent some knowledge of its surrounding, obtained through sensors (such as RGB or depth cameras). However, this knowledge acquisition process is a key challenge. To this end, two major directions can be followed. On the one hand, knowledge can be codified in a sub-symbolic model (e.g., a neural network), which

¹**T. Campari**, L. Lamanna, P. Traverso, L. Serafini, L. Ballan. "Online Learning of Reusable Abstract Models for Object Goal Navigation", *Proc. of IEEE/CVF Computer Vision and Pattern Recognition, New Orleans, 2022 (CVPR 2022)*

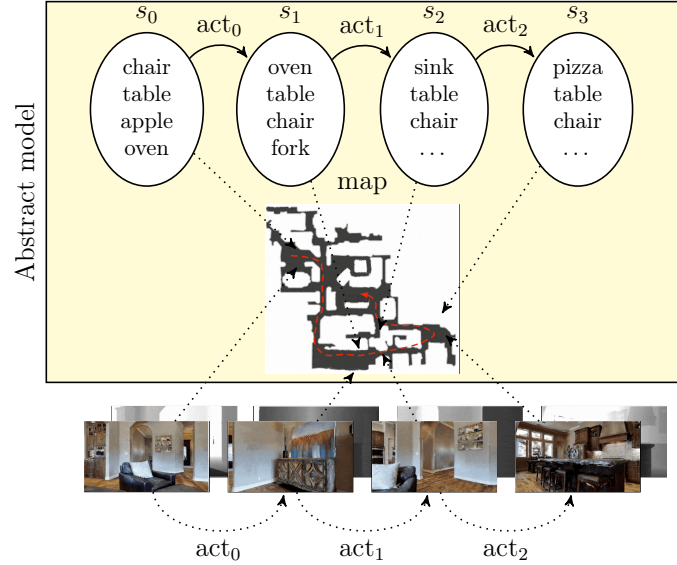


Figure 3.1: During its navigation in a complex 3D scenario, an agent incrementally acquires knowledge about the environment by storing rich semantic information in an Abstract Model. For instance, when the robot is in s_0 , chair and table are visible; by performing act_0 , other objects become visible, thus the Abstract Model is updated to s_1 . Our work shows how knowledge can be incrementally learned and effectively reused over time.

is learned, for instance, by designing supervised or reinforcement learning techniques that can be directly applied to the sensory data [180, 62]. On the other hand, one can adopt a symbolic/semantic representation of the environment (e.g., by exploiting a semantically rich relational structure) which captures the high-level critical aspects of the environment, abstracting away useless details [97, 149].

In our work, we follow this second approach, in the attempt of obtaining a more abstract and general knowledge representation that can be, eventually, reused across time. To this end, an agent, such as a robot navigating in a complex scenario, will represent the acquired knowledge of the environment in an Abstract Model that encodes the following key features: (i) some semantic insights about objects, scene elements, and their relations; e.g., it represents a specific state such as “the agent is close to a fridge and a table is visible from that position” (see Fig. 3.1); (ii) the elements of the Abstract Model are “grounded” to the perceptions; for instance, the agent stores in the Abstract Model some information of each encountered object, such as its position, the corresponding visual features, etc.; (iii) the Abstract Model is dynamically

updated to incorporate the additional information the agent acquires during its operations; i.e., if a new object is discovered, it should be added to the model; *(iv)* the models learned in the past should be reusable by the agent whenever it recognizes to be in an environment that corresponds to the stored model. This last property is essential because it is here that we can observe the maximum utility of the learned Abstract Model.

In this work we specifically focus on the Object Goal Navigation task [19], in which an agent is asked to go close to an instance of a given object class. Recent approaches often tackle this problem by constructing environment’s semantic maps [36, 31] and exploiting SLAM [158, 37]. Instead, we propose to acquire and store the environment knowledge in an abstract, and semantically rich, model. Concretely, such a model is represented by a finite automata whose set of states explicitly describe (at a semantic level) what’s an agent views, given a pose. Thus, a state corresponds to an agent pose, a set of object classes (those visible from that pose), and an estimation of the position of each object. We incrementally learn (online) the Abstract Model by navigating the environment, similarly to [35, 36]. The information associated to each abstract state is obtained from the low-level perceptions, acquired from RGB and RGB-D images processed through segmentation models [83] and the Taskonomy model bank [198]. The learned Abstract Model is then stored for future reuse. Therefore, once an agent recognizes that the current environment is similar to one that it has visited before, the proper Abstract Model representing the information previously acquired within the visited environment can be reloaded by the agent, and then updated with the new observations. To implement this feature, we design a “relocation” mechanism that allows the agent to match states of different Abstract Models. We evaluate our approach on the popular Habitat simulator [145], with 3D real environments from the MatterPort3D dataset [32]. In our experiments we focus on the Object Goal Navigation task, and we show that the Abstract Model is helpful to improve the success rate (e.g., avoiding some false positive detections) and the optimality of the planned path.

Summing up, the contributions of this work are threefold: *(i)* the proposed framework allows an agent to incrementally enhance and reuse previously acquired knowledge, relevant to the current environment; *(ii)* we integrate sub-symbolic techniques such as image processing, path planning, global policy learning, with symbolic reasoning on Abstract Models; *(iii)* our experimental analysis shows that learning and reusing Abstract Models is an effective way to exploit previously acquired knowledge, obtained from noisy observations (e.g., from inaccurate semantic segmentations), for the Object Goal Navigation Task.

3.2 Related Works

Embodied AI. In recent years, several large scale datasets for Embodied AI tasks have been presented, such as Matterport3D [32] and Gibson [185]. These datasets contain 3D reconstruction of environments, which enabled the creation of various photorealistic simulators, such as Habitat [145] or GibsonEnv [185]. Thanks to the new experimental setup offered by these environments, nowadays there are numerous exciting tasks, as described in [6]. Some examples are Point Goal [180] (approach a specific point), Object Goal [19, 31] (approach a specific object), and Vision and Language Navigation [7, 141] (follow instructions in natural language).

In this context, the most common approaches are based on Reinforcement Learning models that exploit policies based on RNNs [113, 98, 144, 62, 120, 27]. For example, [113] solves the point goal task by learning a RL policy, using the A3C algorithm [116] and exploiting auxiliary depth prediction and loop closure classification. [120] proposes a Deep RL framework which uses an LSTM-based policy for Object Goal Navigation. [62] proposes Scene Memory Transformer, an attention-based policy [170] that can exploit the least recent steps performed by the agent. In this case, the policy training is performed using the Deep Q-Learning algorithm [115]. Starting from the work done in [198], [146] shows that Mid-Level Vision produces policies that learn faster and generalize better w.r.t. learning from scratch, especially for the point goal navigation task. Previously described approaches require task specific end-to-end training with RL. In contrast our approach is more general, since we train a RL policy to maximize the environment exploration, and can be used to solve different tasks.

Recent works [35, 36, 31] exploit explicit maps constructed from images. Notably, [35] proposes Active Neural SLAM (ANS) which constructs an obstacle map from depth observations. An RL algorithm is then applied on such a map, with the objective of learning a global policy that selects a point, reached via path planning, to maximize the environment exploration. An extension is then proposed in [36], where the occupancy map is enriched with semantic information about objects in the scene. The Global Policy, trained specifically for the Object Goal Navigation task, exploits the semantic information available in the map. Using semantic maps to store information about the environment for future reuse, which is our main objective, requires relocation algorithms, which are not considered in previous approaches. In our approach, instead of relying on relocation w.r.t. semantic maps, we store information of previously visited environments in an Abstract Model, where each state is associated with some visual features sufficient for relocation.

Finally, [31] proposes a new method to construct semantic maps that exploit an encoder-decoder model with a Spatial Memory Transformer. The generated semantic maps are then tested for the Object Goal Navigation task. Furthermore, they propose also an experiment in which pre-computed maps are reused in the Object Goal Navigation task. However, this approach assumes perfect relocation (i.e. the absolute position of the robot is provided).

Learning Abstract Models. Abstract Models learning in planning has the objective of inducing an Abstract Model that describes actions, starting from sequences of observations about their execution. [51, 71] propose to learn action models in a structured language starting from complete observations. [119] learn action models from noisy and incomplete observations. [202] learn an action model on a target domain by transfer learning from a set of source domains and by partial observations. [4] propose a method for learning action models from either complete or incomplete observations. The work by [24], provides a framework for learning first-order symbolic representations from complete information about action execution. In all these approaches, learning is performed from symbolic observations, and sensory perceptions in a continuous environment are not considered. In this work, we also tackle the problem of abstracting the sensory perceptions into a finite set of states.

Alternative approaches learn a discrete Abstract Model from continuous observations. Causal InfoGAN [94] learns discrete or continuous models from high dimensional sequential observations. This approach fixes a priori the size of the discrete domain model. LatPlan [12, 11] takes in input pairs of high dimensional raw data (e.g., images) corresponding to transitions and learn an action model. LatPlan is an offline approach, while our approach instead learns online and without fixing the dimension of the Abstract Model. The work by [97] proposes an online method to learn Abstract Models by mapping continuous perceptions to deterministic state transition systems. With respect to our approach, they requires an input draft Abstract Model, and do not deal with complex perceptions like RGB-D images.

Our approach shares some similarities with the work on planning by reinforcement learning [84, 191, 122, 142, 66], since we learn by acting in the environment. However, these works focus on learning policies and assume the set of states and the correspondence between continuous observations from sensors and states are fixed.

Cognitive architectures At the cross border between AI, Robotics and Cognitive Science, many systems were developed [171, 101, 91]. Systems like SOAR [96, 95], CRAM [20], or ISAC [87] provide useful architectural elements

to generalize and distinguish between the type of memories and mechanisms used. Furthermore, they present insights about different uses of the symbolic commonsense knowledge. However, in this work, the background knowledge is not directly injected, but is learned and extended through different episodes and actions.

3.3 Preliminaries

Abstract Model. The knowledge of the agent about an environment is represented by a finite state machine in which each states is associated to the “corresponding” visual features. Formally, a finite state machine is a triple $\mathcal{D} = \langle S, A, \delta \rangle$ where S is a finite set of states, A is the set of actions that the agent can execute, and δ is a deterministic transition function among states, i.e, a function $\delta : S \times A \rightarrow S$. Each state $s \in S$ is associated to a triple $\langle \mathcal{F}_s, \mathcal{C}_s, \{\mathcal{F}_{s,c}\}_{c \in \mathcal{C}_s} \rangle$ where \mathcal{F}_s is a set of numeric features associated to the state (i.e. a feature vector extracted from the RGB image); \mathcal{C}_s is a finite set of identifiers for the objects visible by the agent in the state s ; for all $c \in \mathcal{C}_s$, $\mathcal{F}_{s,c}$ is a set of real features associated to the view of the object c in the state s (it includes for instance the estimated relative position, the bounding box and a set of visual features).

Since the agent is aware of different environments, it keeps multiple Abstract Models $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(n)}$. We don’t assume a one-to-one correspondence between models and environments, since the agent might associate different models to the same environment. For example, the agent could erroneously build two models for the same environment because the second time it enters in the environment, it does not realize that this has been already visited.

3.4 Object Goal Navigation

In the Object Goal Navigation task [144], an agent is required to go close to an object of a specific class (such as *fridge* or *bed*) – referred as to *object goal* – starting from a random position within an unknown and static environment, in less than 500 actions. A particular instance (“run”) of this task is called *episode*. To reach the object goal, the agent is allowed to execute a set of actions (also called *steps*): namely, `move_forward` (by *25cm*), `turn_left`, `turn_right` (by *30°*), `stop`. At every step, the agent can observe the environment via a set of sensors providing an RGB-D image and the agent pose $\langle x, y, \theta \rangle$, relative to the initial one (which is $\langle 0, 0, 0 \rangle$). The agent ends an episode by executing the `stop` action; if its distance from the closest object goal is less than a

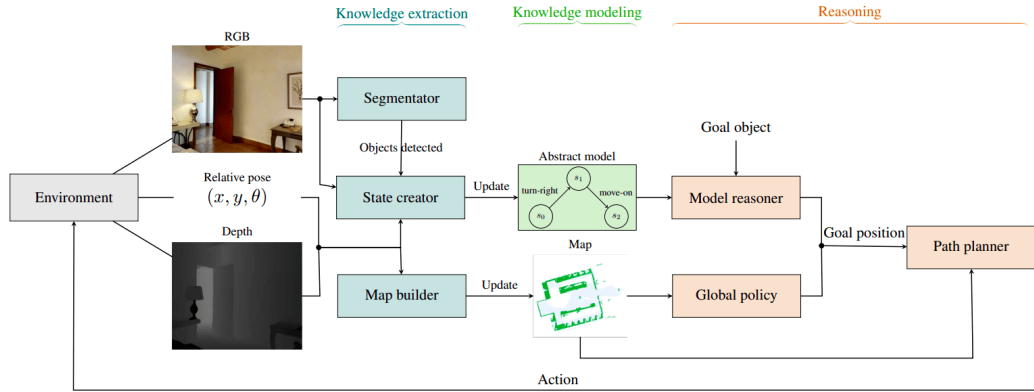


Figure 3.2: Overview of the proposed architecture for Object Goal Navigation.

threshold (set to $1m$), the episode is considered a success, otherwise a failure. The solution of the task involves multiple challenges. Firstly, the agent must explore the environment in an effective way by exploiting SLAM techniques to learn a map of the environment. Then, it has to recognize new objects in the environment whenever they are into its current view, through object detection. Finally, it must be able to approach the goal objects, by using path planning algorithm to decide which actions it has to execute.

In the standard task each episode is independent from the other and no information is transferred across episodes. We call it *memory-less* setting. We also introduce (and test in 3) a new setting called *with-memory*, where the agent can exploit the knowledge acquired in previous episodes. In particular, if the agent realizes that is visiting an already visited environment, it can retrieve and exploit the knowledge previously acquired. We believe that the *with-memory* setting is much closer to real scenarios, where an agent should accumulate and reuse previously accumulated knowledge. Notice that the *with-memory* setup introduces new challenges, concerning how and which previously acquired knowledge can be reused in the current situation (relocation and aggregation). Furthermore, in the *with-memory* setup, dealing with previously acquired noisy knowledge is even more challenging, due to the errors accumulation.

3.5 Approach

An overview of the main cycle executed at every step by the agent to reach an object goal G is shown in Figure 3.2. This cycle is composed of three main phases: 1) *Knowledge extraction*, 2) *Knowledge Modeling*, and 3) *Reasoning*. The approach extends [35, 36] by allowing the agent to learn Abstract Models

and reuse them.

Knowledge Extraction. At every iteration the agent perception is composed of the RGB-D image, corresponding to the egocentric view of the agent, and the agent current relative pose. The Segmentator module [83] extracts object segmentations from the RGB-D image. The Map Builder module creates an egocentric map with a classical SLAM approach [73] from the current depth image and pose. Finally, the State Creator module generates an abstract state $s = \langle \mathcal{F}_s, \mathcal{C}_s, \{\mathcal{F}_{s,c}\}_{c \in \mathcal{C}_s} \rangle$ where: \mathcal{F}_s is the set of state features extracted from the RGB image by the auto-encoder of the Taskonomy model bank [198]; \mathcal{C}_s are the object classes extracted by the Segmentator module; for all $c \in \mathcal{C}_s$, $\mathcal{F}_{s,c}$ contains: the position on the map, the bounding box, and the distance from the agent of every visible object of type c . The object position is estimated by adding the depth value of the bounding box centroid to the agent pose.

Knowledge Modeling. In the Knowledge Modeling phase the environment map and the current Abstract Model are updated with the knowledge extracted in the previous phase. Namely: the current map is extended with the additional information present in the egocentric map, and the state s extracted by the State Creator is added to the current Abstract Model if not present. Finally, the transition function is extended with (s_{prev}, a, s) , where s_{prev} is the previous state, and a is the last executed action.

In the *with-memory* setting, if s matches with a state in a previously learned Abstract Model, this model is reloaded and merged with the current one, using the procedure described in Section 3.5.1.

Reasoning. In the Reasoning phase, given the object goal class G , the agent looks if the current Abstract Model contains a state with an object of type G (i.e., $\exists s \in S$ s.t. $G \in \mathcal{C}_s$). In such a case, the agent selects one object of type G and sets the position of the object as goal point on the map. If the Abstract Model contains multiple states with objects of type G , then the agent ranks these objects according to the number of states from which they are visible, and selects the closest one among the top five. We prefer mostly seen objects in order to mitigate the errors of the Segmentator. Indeed, the more points of view from which an object is detected, the less probable that it is a false positive of the Segmentator. Alternatively, if the Abstract Model does not contain any state with G (e.g., in exploring a new environment, the agent might not have seen any object of type G), a goal position is computed by our Global Policy, based on [35, 36]. Namely, given the current map,

the policy seeks for a position on the map that maximizes the environment exploration. Once the target position is set, either by the reasoner or by the Global Policy, the agent computes a plan with a path planner, based on the Fast Marching algorithm [151], to reach the target position, and executes the first action. To compute a plan, all unexplored areas of the map are considered navigable; this enables the agent to discover new scene elements and objects, thus enriching both the map and the Abstract Model.

3.5.1 Abstract Model Reuse

In the *with-memory* setting the Abstract Model learned at each episode is stored by the agent for future re-use. Therefore, the knowledge of the agent is constituted by n Abstract Models $\{\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(n)}\}$. When the agent starts a new episode, it initializes a new Abstract Model $\mathcal{D}^{(n+1)}$. At every step of this episode, the agent looks if its current state $s = \langle \mathcal{F}_s, \mathcal{C}_s, \{\mathcal{F}_{s,c}\}_{c \in \mathcal{C}_s} \rangle$ matches a state in $\{\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(n)}\}$. Matches between states of different Abstract Models are computed by the cosine distance among the state features \mathcal{F}_s . Therefore the best match is computed as follows:

$$s^* = \underset{\substack{s^{(i)} \in S^{(i)} \\ i \in 1, \dots, n}}{\operatorname{argmin}} \cos_dist(\mathcal{F}_s, \mathcal{F}_{s^{(i)}})$$

where $S^{(i)}$ is the set of states of the i -th model $\mathcal{D}^{(i)}$. If $\cos_dist(s, s^*)$ is lower than a given threshold (an hyperparameter of our model), and s^* is a state of the i -th Abstract Model, then $\mathcal{D}^{(n+1)}$ is merged into $\mathcal{D}^{(i)}$ and the resulting one $\mathcal{D}^{(i)}$ is considered as the current Abstract Model. The resulting model contains all states of the two merged models, and the knowledge is incrementally enhanced through episodes. After such a merging the agent does not look for further matching in the current episode.

Notice that the match could not be perfect since the poses of the robot in the matched states s and s^* might be slightly different. This matching difference can propagate to the object positions recorded in the Abstract Model, thus the agent can rely on wrong information. To prevent these potential errors, we propose two different strategies: namely *hard* and *soft*. In the *hard* strategy, we assume that the matching is always perfect and the agent blindly believes in the matched Abstract Model, i.e., it goes to the goal object position returned by the Model reasoner without looking for other goal objects on its path. In the *soft* strategy, the agent tries to mitigate the effects of non-perfect matches. To do this, the agent looks for the goal object in the area around the goal object position given by \mathcal{D} . The dimension of the area around the goal object is proportional to the distance among the agent

position in the matching state and the goal object one. Moreover, during its path, it continuously looks for a goal object, possibly terminating the episode before reaching the area around the goal object position suggested by the Abstract Model.

3.6 Implementation Details

We used the Habitat Simulator [145] with the Matterport3D dataset [32], which contains 90 different scenes with a total of 194K RGB-D images. Habitat allows to simulate the navigation in these 90 different scenes.

The Global Policy, which select the exploration goal, is trained for 10M steps on the 56 training scenes of Matterport3D (50 as training and 6 as validation) using the Proximal Policy Optimization (PPO) RL algorithm. The Global Policy consists of 5 Conv Layers with the ReLU activations and MaxPooling2D as in [35, 36]. For the Semantic Segmentation, we used the RedNet model [83] pretrained on the 40 classes of Matterport3D (check [194] for details about the performance in the OGN task). The features extractor, which computes \mathcal{F}_s from the RGB data, is the encoder of the Taskonomy model bank [198], and \mathcal{F}_s is a vector with dimension 2048. The cosine distance threshold for states matching is set to 0.3. The SLAM algorithm, which computes the egocentric map, is based on [73].

In our experiments, we evaluate our approach on the OGN task. Our aim is to show that the *reuse* of previously acquired knowledge, in the form of Abstract Models, can enhance the navigation in existing approaches. Furthermore, we empirically demonstrate our claims with a limitation and failure analysis and a qualitative comparison about reusing *vs* not reusing previously acquired knowledge.

3.6.1 Evaluation Metrics

The OGN task is evaluated with four standard metrics: the *Success Rate*, the *Success weighted by Path Length (SPL)*, the *SoftSPL*, and the *Distance To Success (DTS)*.

The *Success Rate* is defined as the ratio between the successful and the total number of episodes.

The *SPL* [6] estimates the efficiency of the agent in reaching the goals, and it is defined as:

$$\text{SPL} = \frac{1}{N} \sum_{i=1}^N S_i \frac{l_i}{\max(p_i, l_i)}$$

where N is the number of episodes, l_i is the shortest-path distance from the agent’s starting position to the closest goal point in an episode i , p_i is the length of the path followed by the agent in the episode i , and S_i is a boolean success indicator of the i -th episode.

The *SoftSPL* [31] is similar to the *SPL*, but measures the path optimality in all the episodes, without penalizing the unsuccessful ones with a zero score; it is defined as:

$$\text{SoftSPL} = \frac{1}{N} \sum_{i=1}^N \left(1 - \frac{d_{T_i}}{d_{init_i}} \right) \frac{l_i}{\max(p_i, l_i)}$$

where d_{init_i} is the geodesic distance between the initial position of the agent and the target point, and d_{T_i} is the geodesic distance between the final position of the agent and the goal point. Both refers to the i -th episode.

Finally, the *DTS* measures the mean distance from the closest goal point, mathematically:

$$\text{DTS} = \frac{1}{N} \sum_{i=0}^N \max(\|x_i - g_i\|_2 - d, 0)$$

where $\|x_i - g_i\|_2$ computes the L2 distance for the i -th episode and d is the success threshold ($1m$).

3.7 Experiments

3.7.1 Reusing Abstract Models

Here, we report our investigation on different way of reusing knowledge and their advantages. To this aim we have developed the following four models:

Active Neural SLAM (ANS*): it is our implementation of the ANS model[35]

². This constitutes our baseline, since it does not exploit any previously acquired knowledge.

Hard Pre-explored (ANS*+HP): it is the simplest extension of **ANS***, based on our approach (Section 3.5). Firstly, the agent is provided with an Abstract Model for each environment. Then these Abstract Models

²We checked the coherence of our implementation by running the same experiments as in [35]; the performance are comparable: namely **ANS [35]** reports 7.056, 0.321 and 0.119 of DTS, Success, and SPL, respectively; our implementation obtained 6.721, 0.313 and 0.127 on the same metrics.

are initialized by performing 10000 exploration steps; for every episode the agent can reuse one of the pre-acquired Abstract Models, using the Hard strategy.

Soft Pre-explored (ANS*+SP): similar to **ANS*+HP**, but the agent reuses the provided Abstract Models by applying the Soft strategy (see Section 3.5.1).

Soft Incremental (ANS*+SI): this is our “full model”. In the first episode, the agent is provided with no Abstract Models; then, in the consecutive steps, the agent can reuse and incrementally extend all the Abstract Models learned in previous episodes, using the Soft strategy.

The fundamental difference between **ANS*** / **ANS*+SI** and **ANS*+HP** / **ANS*+SP** is that the formers do not take pre-acquired knowledge, while the latters require such a knowledge. Moreover, **ANS*+SI** is the only model in which the agent extends the Abstract Models with the additional knowledge acquired through episodes. Finally, all the versions but **ANS*** use the *with-memory* setup described in 3.4.

Table 3.1 shows the results of our variants on the validation set of Matterport3D, composed of 2195 episodes in 11 environments. This is a standard benchmark for the OGN task [36, 31]. **ANS*+HP** achieves higher results than **ANS***, as expected, since **ANS*+HP** is provided with additional input knowledge. Furthermore, **ANS*+SP** obtains better results than **ANS*+HP**, due to the fact that the Soft strategy mitigates the errors introduced by the matching of Abstract Models in different episodes (Section 3.5.1). Remarkably, **ANS*+SI** outperforms all the other versions, providing a relative improvement of +8.13% in *success* and +11.9% in *SPL* w.r.t. **ANS*+SP**. From the results of Table 3.1, we can deduce that the incremental learning of Abstract Models is more effective than providing the agent with the pre-acquired input Abstract Models.

Furthermore, the fact that the agent starts in each episode from a different location allows the **ANS*+SI** variant to cover spaces inside the environment that are hardly reachable with a single long pre-exploration. **ANS*+SI** is able to relocate the agent in 69.7% of the episodes, confirming that the method exploited the matching system.

3.7.2 Effects of Knowledge Accumulation

This section aims to investigate how accumulating knowledge in the Abstract Models affects the agent performance. To do this experiment, we need to limit the quantity of noise recorded in the Abstract Model (e.g., the false

Method	DTS↓	Success↑	SPL↑	SoftSPL↑
ANS*	6.417	0.240	0.102	0.191
ANS*+HP	6.352	0.251	0.105	0.206
ANS*+SP	6.294	0.258	0.117	0.214
ANS*+SI	6.155	0.279	0.131	0.233

Table 3.1: Results achieved by our variants on the Matterport3D validation set.

Method	DTS↓	Success↑	SPL↑
ANS*	6.721	0.313	0.127
ANS*+SI	6.347	0.354	0.150

Table 3.2: Results obtained on a subset of the validation set of Matterport3D, containing object classes which are in both MS-COCO and MatterPort3D datasets (658 episodes across 11 environments).

positives given from the Semantic Segmentator). Hence, we selected a subset of the Matterport3D validation set on which the Semantic Segmentator is more stable. This subset was built as in [36] and contains episodes with a goal object in one of the following classes: *chair*, *sofa*, *plant*, *bed*, *toilet*, *tv*, *table*, and *sink*.

Table 3.2 reports the results obtained with **ANS*** and **ANS*+SI**. Our results show that reusing Abstract Models (**ANS*+SI**) allows the agent to take better paths (15% SPL) and brings the agent closer to the goal object (6.34m DTS) compared to the version without knowledge reusage. The analysis on the evolution through episodes of the success rate is visible in Figure 3.3. Here, we plotted in dashed line the average success rate for each episode across the 11 environments (e.g., point 0 represents the average success rate across the 11 environments in their first episode). The solid thick line represents the moving average of the success rate with a window size of 5. From the figure, it is clear that accumulating knowledge over episodes consistently enhances the success rate.

3.7.3 Semantic Maps and Abstract Models

Another method to represent knowledge about the environments are Semantic Maps [31]. Semantic maps are obstacle map enriched with information about object classes. Here, we compare systems that use Semantic Maps with our method, and investigate how semantic maps and Abstract Models can be

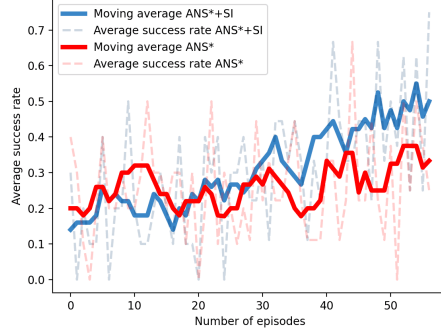


Figure 3.3: A plot of the moving average success rate in the ANS*+SI model. The window size was set to 5.

Method	DTS↓	Success↑	SPL↑	SoftSPL↑
SMNet [31]	7.316	0.096	0.057	0.087
SMNet (GT)	5.658	0.312	0.207	0.282
ANS*+SI	6.155	0.279	0.131	0.233
SemExp*+SI	5.785	0.347	0.151	0.274

Table 3.3: Results obtained on the validation set of the Matterport3D dataset (2195 episodes across 11 environments). Note that **SMNet (GT)**, as explained in [31], exploits ground truth free space maps extracted directly from the Habitat API.

suitably combined.

A method for OGN that exploits a precomputed semantic map is **SMNet** [31]. In **SMNet**, the plan to reach an object goal G is obtained by computing the shortest path to an object of type G . The method makes the simplifying assumption that the absolute position of the agent is known, therefore re-location is not needed. [31] also considers a version, called **SMNet(GT)** that assumes ground truth free space maps. Another system that exploits semantic maps is **SemExp** [36]. **SemExp** is an evolution of **ANS** where the base architecture is the same, but the Global Policy takes semantic maps in input. Such a policy seeks to directly find the object goal, instead of maximizing the environment exploration. To understand how semantic maps and Abstract Models can be combined, we integrate our **SI** approach on top of **SemExp**. This version is called **SemExp*+SI**. Table 3.3 compares all this different versions of reusing knowledge. We used the same split used in [31], which is the validation set of the Matterport3D dataset (2195 episodes across 11 environment). Remarkably, **ANS*+SI** and **SemExp*+SI** outperform **SMNet**

by large margin. Furthermore, the Global Policy exploited in **SemExp*+SI** increases all the metrics w.r.t. **ANS*+SI** counterparts. The higher *success* rate of **SemExp*+SI** w.r.t. **ANS*+SI** (+6.8%) suggests that the way in which the environments are explored plays a crucial role in how the Abstract Models are learned. It’s interesting to notice that **SemExp*+SI** has similar performances to **SMNet (GT)**, that exploits pre-computed maps with ground truth free space.

3.7.4 Limitations and Failure analysis

One of the major limitations in our model comes from the abstraction of input sensory data. The output of the Semantic Segmentator, as well as the visual features stored in the memory, could be affected by errors. This could lead to semantic drift and put a bound on the quality of the knowledge representation. Furthermore, in the *without-memory* setup, the Abstract Model does not provide a significant added value with respect to simpler representations (such as semantic maps). However, this is not the case in the *with-memory* setting, in which the abstraction encoded in the Abstract Models is a cornerstone for the reuse previously acquired knowledge.

Hereafter, we report a failure analysis aiming at understanding why the agent fails in the **SI** setting. In particular, we try to understand quantitatively how the errors introduced by the Semantic Segmentator affect the reliability of the Abstract Models. To this end, we randomly sampled 200 failed episodes in which the Abstract Model was reloaded from the experiment in Table 3.2. Then we annotated the failures w.r.t. five classes: (i) **Last Mile (Navigation Failure)**: the agent correctly navigated to an instance of the goal object but was not able to reach it ($DTS < 2m$); (ii) **Hallucination (Abstract Model Failure)**: the agent approached the goal point extracted from the Abstract Model, but there was no goal object occurrence nearby the suggested location; (iii) **Detection (Sensors Failure)**: the agent, during its path to the goal point suggested by the Abstract Model, found a wrong instance of the goal object and approached it; (iv) **Exploration (Abstract Model Incompleteness)**: the reloaded Abstract Model has no information about possible goal object locations, and the agent could not find any instance in 500 steps; (v) **Misc**: the agent reloaded the Abstract Model but has a generic failure (e.g., agent is trapped by reconstruction debris at spawn).

Notably, we have two possible failure cases directly linked to the reloaded Abstract Model: Exploration, where the agent has not enough information about the environment, and Hallucination, where the agent relies on wrong information. In Figure 3.4 are reported the statistics about the failed episodes. We can observe how the Abstract Model generated by **SemExp*+SI** gave

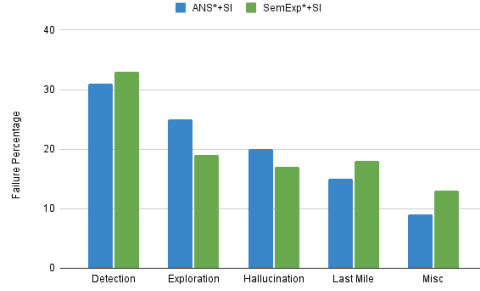


Figure 3.4: Barplot of failure cases for **ANS*+SI** and **SemExp*+SI** models on the validation set of Matterport3D.

fewer failures wrt **ANS*+SI** in the Exploration and Hallucination failures, highlighting how the different Global Policy affects the creation of the Abstract Models. Furthermore, the majority of the failures are in Detection for both models. This also suggests that a better reconstruction of the Semantic Segmentations can improve a large margin of performance.

3.7.5 Qualitative examples

In Figure 3.5 and 3.6, we report a qualitative comparison among **ANS*** and **ANS*+SI** in the same episode of the scene *2azQ1b91cZZ* provided in the MatterPort3D dataset.

ANS* starts by exploring the environment, but it never encounters an instance of the Sofa object class. Furthermore, the exploration leads the agent very far from the nearest sofa, more than 10 meters. This is because the environment of the scene *2azQ1b91cZZ* is very large w.r.t. the average dimension of other environments, and the agent is likely to take paths towards areas far from the goal object ones. Therefore, with a limited number of 500 steps, the agent cannot easily find an object when navigating into the environment for the first time. Figure 3.6, shows the same episode with the exploitation of the incrementally learned Abstract Model. Particularly, at step 15, the agent’s state matches a state of the previously learned Abstract Model that, therefore, is reused. The agent’s coordinates system is rescaled to match the reused domain one, and the goal point is chosen. Subsequently, starting from the selected goal object position, is created an exploration area that the agent must explore to find the goal object. At step 82, the agent has reached the selected area and consequently approaches the sofa it was looking for.

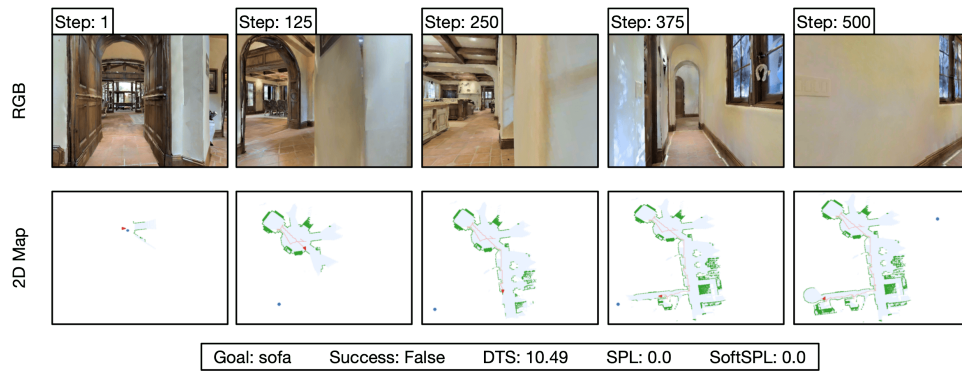


Figure 3.5: A failed episode with the **ANS*** variant. The agent explored the environment for 500 steps without finding any Sofa occurrence. Green pixels on the map are the obstacles, light blue pixels are explored areas, and the blue point is the goal position.

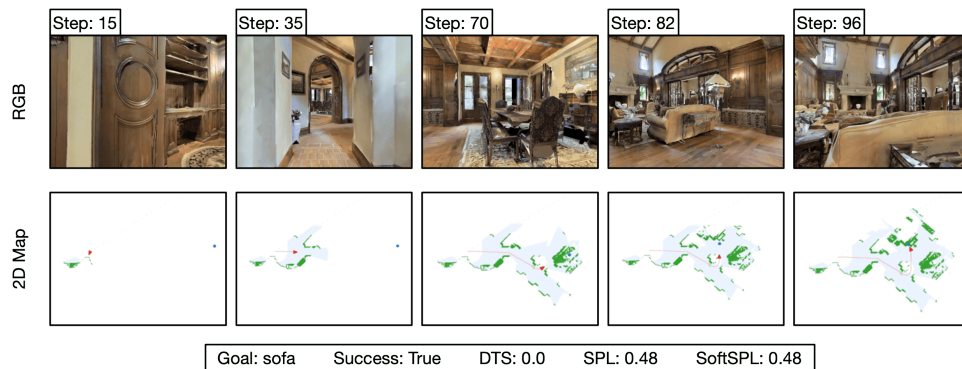


Figure 3.6: A success with **ANS*+SI** variant. The agent correctly matched its current state with one in the domain and used the information stored to navigate towards a Sofa successfully.

3.8 Conclusion

This chapter presents a novel approach that allows an agent (*i*) to incrementally acquire and store knowledge about a set of unknown environments, and (*ii*) to reuse the acquired knowledge, represented as an Abstract Model, when the agent returns to an already visited state. We evaluate the proposed method on the Object Goal Navigation task. Our experiments show that reusing Abstract Models is effective. An ablation study on different strategies to reuse such knowledge confirmed that incremental learning works better than reusing Abstract Models learned offline. The failure analysis highlights that reusing Abstract Model does not constitute the major reason of failure, which is mostly ascribable to the Semantic Segmentation module. The qualitative analysis highlights how the effect of reusing Abstract Models affect the agent’s behaviour. Future works will focus on integrating more semantic information about the environment (e.g., correlation between room types and object types which are present in the room) and tackling other Embodied AI tasks that require reasoning at a symbolic level (e.g., Image Goal Navigation [201], Embodied QA [56], and Rearrangement [18]).

Chapter 4

Modular Multi-Object Navigation

In this chapter, we continue to explore how it is possible to effectively reuse previously acquired knowledge for an Embodied AI (**R2**). Specifically, we propose an agent that records acquired knowledge in a semantic map and reuses it in long-horizon tasks. The agent is tested in the context of Multi-Object Goal Navigation (MultiON), where an agent is provided with n consecutive goals. The current goal is revealed only when the previous is reached, thus the agent is not aware of what is specifically useful to record and reuse. For this reason, MultiON is a great testbed for the efficient reuse of acquired knowledge. Our agent has a modular architecture, that exploits a pretrained PointGoal Navigation model, an exploration strategy and a map to record acquired information.

4.1 Introduction

¹Embodied AI is rapidly gaining interest in deploying computer vision systems into realistic environments to solve interactive tasks. Visual embodied agents can operate in photo-realistic environments, often scans of real-world houses, and enact action to meaningfully affect the state of the environment. Several milestones have led to tremendous research progress and increasing performance across tasks. These include fast and high-fidelity simulators [145, 162, 59], deep reinforcement learning advancements [147, 79], memory representation [78, 31, 176], self-supervision [44, 77, 150, 125], and parallel infrastructure [180, 61, 103, 14]. Now that embodied agents are highly skilled at basic navigation with simplifying assumptions [145, 180, 195], as a natural next step, more realistic and challenging tasks have been pro-

¹S. Raychaudhuri, **T. Campari**, U. Jain, M. Savva, A. X. Chang. "Modular-MON: Modular Multi-Object Navigation". Under submission.

posed [18, 178, 58]. Although embodied agents now have the essential skills to

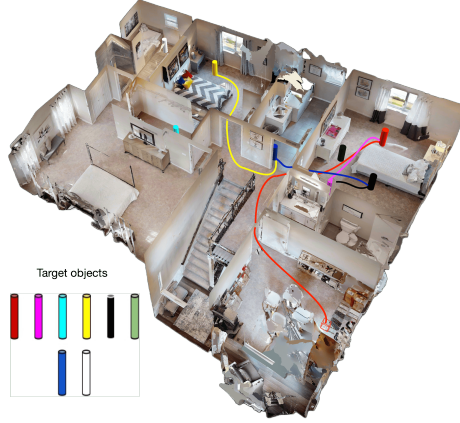


Figure 4.1: Example 5ON episode on an HM3D dataset scene. The agent needs to find, in order: red, pink, black, blue, and yellow cylinders. Another three cylinders are distractors in the scene.

navigate to specific coordinates or point-goals, semantic navigation tasks are far from being solved. These tasks entail finding objects or areas in the environment based on semantic labels or visual goal specification [189, 5, 177, 74], inherently increasing the task complexity. Other challenges include long-horizon navigation as investigated through multi-hop semantic navigation (MultiON) [176]. The objective is to navigate to an (episode-specific) ordered list of objects. In long-term reasoning tasks like MultiON, agent policies must efficiently use mapping, exploration, and navigation skills. This is particularly challenging for an agent due to the inherent partial-observability of the environment, which demands both exploratory skill, for when the object is not yet visible, and exploitative skill once the object has been located.

However we humans can efficiently plan when given long-horizon tasks. We can walk around searching for objects and remember them in order to return at a later point. Drawing from such intuitions, our structure approach has four modules dedicated to: (a) *object detection* – to parse objects in the raw visual of the agent, (b) *map building* – a semantic map storing observed objects for easy querying, (c) *exploration* – to utilize learned priors for efficient search when an object of interest is yet to be located, and (d) *navigation* – to reach an target object that has been located. The first two contribute to parsing and storing semantic knowledge about the environment, while the latter two enable efficient embodied navigation.

We conduct all our experiments on MultiON 2.0, a new version of the MultiON dataset [176] built on top of the large-scale dataset HM3D [135].

Compared to MultiON, this new version contains up to 5 target objects (5ON task) and includes distractor objects, which are helpful to test agents’ ability to distinguish between different targets. Moreover, we also added another variant where we add *Natural objects* instead of cylinders; this modification should add more realism to the task compared to the cylinders used in the previous dataset version. Furthermore, this dataset contains 10x more environments compared to the standard MultiON. This dataset is a step towards the recent direction taken by the embodied AI community, where to close the sim2real gap many works are proposing large-scale datasets, like HM3D or ProcTHOR [59] that can serve as pretraining for robotic agent deployment.

Our experiments show that having a modular approach with a simple random exploration policy and a pretrained PointGoal navigation policy is effective in the complex task of MultiON. In summary, we make the following contributions: i) we create MultiON 2.0, a new large-scale dataset for multi-object navigation; ii) we show that a pretrained PointNav outperforms analytical path planners by a significant margin; and iii) we compare rule-based exploration strategies and find that a simple random strategy outperforms more complex methods.

4.2 Related Work

We focus on the MultiON task, a popular Embodied AI task, by proposing a modular architecture that relies on a pretrained policy on a PointGoal Navigation task and a simple rule-based exploration strategy. In the following, we will report the most relevant literature.

Embodied AI Tasks. The availability of large-scale datasets such as Matterport3D [32], Gibson [185], and Habitat-Matterport3D (HM3D) [135] along with photo-realistic simulators, such as Habitat [145], GibsonEnv [185], AI2-THOR [90] etc. have enabled a diverse set of tasks for embodied agents. These include *point-goal navigation* [145, 180, 195], where the target is a single point, *object-goal navigation* [19, 194, 36, 89], where the target is a semantic label of an object, and those of *instruction following* [7, 93, 114, 42], where the agent needs to follow instructions in natural language. Most relevant to our work is the task of *multi-object navigation* (MultiON). It is a generalized extension of the object-goal navigation task, where the agent must reach multiple objects in a sequence instead of a single object in object-goal navigation.

Modular Navigation in Robotic Vision. A common paradigm in classical robotics is to divide navigation into mapping (via simultaneous localization and mapping [65]) and path planning (by computing path to goal locations [86, 151]). Taking inspiration from this vision along with the success of deep nets, a hybrid approach of neural high-level policies with model predictive control emerged as a more robust and sample-efficient alternative for navigation [17, 85]. A similar trend was seen in the aforementioned embodied AI tasks with first wave of approaches being reactive or recurrent deep net policies that were largely monolithic [73, 88, 145, 39, 82]. Banking on interpretable structure of neural modular networks [8, 92] in vision, modular policies for navigation consisting of separately trained modules and using auxiliary data have shown to be significantly more sample-efficient. For instance, Chaplot et al. [35] achieved better performance on point-goal navigation with just 10Mn frames of training instead of 75Mn frames. The central theme is “why (re-)learn skills that are analytically or heuristically already solved”. In the embodied AI context participating entries in the very challenging MultiON 2022 competition [58] are mostly modular, combining learned and heuristic/rule-based modules to achieve superior performance.

Exploration in Navigation. Exploration has been studied extensively in both visual navigation and robotics and it is particularly critical for a long-horizon agent in the context of semantic navigation tasks. A common approach is to estimate an exploratory waypoint or goal and navigate towards it [17, 137]. To select an exploration goal, traditional methods explore the environment based on heuristics, such as a frontier [190] point between explored and unexplored regions. More recent works propose learning-based methods so as to generalize to unseen environments better. Notable works include learning end-to-end RL exploration policies from coverage rewards [43, 134, 132] and intrinsic rewards using inverse dynamics [125, 126]. Modular and geometric approaches have also maximized coverage on metric maps by using first-person depth images [35], predicting semantic maps [36], and topologically-structured maps [37, 74, 177]. Very recently, Luo et al. [107] proposes a simple rule-based exploration strategy called ‘Stubborn’, which outperforms state-of-the-art methods in the ObjectNav task. The agent assumes a local grid around it and selects each of the four corners, in turn, as the exploration goal. This agent is ‘stubborn’ because it keeps moving towards the goal until it collides. In this work, we explore a few variants of Stubborn and compare them with other rule-based methods.

	#Tr	#Ep.	#Val	#Ep.	#Test	#Ep.
MultiON [176]	61	3.05M	11	1050	18	1050
MultiON 2.0	800	8.00M	30	1050	70	1050

Table 4.1: **Comparing dataset statistics.** #[Tr, Val, Test] is the number of scenes respectively in the training, validation, and test splits. #Ep. is the number of episodes in each split. We created MultiON 2.0, for 1ON to 5ON, with the same number of episodes.

Transfer Learning in Embodied AI. While transfer learning is studied extensively across datasets of computer vision, fewer works have investigated it in the context of Embodied AI. Shen et al. [154] shows that by pre-training on vision tasks, the agent learns the target task better. Lin et al. [102] shows that pre-training improves the downstream manipulation task. Some works [38, 175] focus on a multi-task learning setup that enables sharing knowledge between multiple tasks. Al-Halah et al. [5] pre-train a source policy on image-goal navigation and show transfer using pluggable goal encoders for other definitions of semantic visual navigation. Very few works in the ObjectNav task have tried to leverage an off-the-shelf PointNav model as their navigation policy. For example Georgakis et al. [67] uses a pre-trained PointNav model as their local policy while predicting semantic maps outside the agent field of view. Similarly, as our navigation policy, we re-use a PointNav policy that was trained on HM3D scenes [135]. We show that it outperforms analytical path planners by piggybacking on the near-perfect performance of point-goal navigation.

4.3 MultiON 2.0 Dataset

With this work, we release MultiON 2.0 – a large-scale dataset for the Multi-Object Navigation task. In this section, we include details about objects that are inserted as goals for Multi-Object Navigation. We also include necessary design choices for procedural generation of episodes and associated data statistics.

Diversity of objects. To supplement the existing data [176] we include natural objects beyond synthetic cylindrical goals. Particularly, Wani et al. [176] inserted eight cylinder-shaped objects of different colors (black, blue, cyan, green, pink, red, white, and yellow) in the Matterport3D [32] scenes,

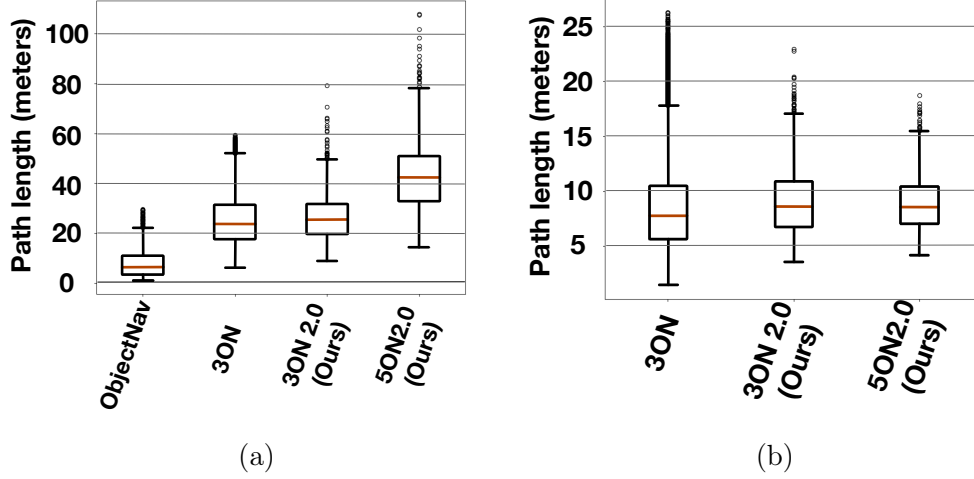


Figure 4.2: **Comparing path lengths across tasks.** We include boxplots for (a) average episode length across {ObjectNav [19], 3ON [176], 3ON 2.0 (ours), and 5ON 2.0 (ours)} datasets; (b) average geodesic distance between successive objects across different datasets.

each 0.75m in height and 0.05m in radius. We increase the task difficulty by requiring better visual understanding in agent policies. To this end, along with a version of the dataset with cylinder objects (*CYL-objects*), we include realistic-looking objects that can naturally occur in houses. In line with prior work, we insert eight objects: backpack, basketball, electric piano, guitar, rocking horse, teddy bear, toy train, and trolley bag. We choose objects that are large and visually diverse for them to be detected and identified by embodied agents. We call this set of objects *Natural-objects* (NAT-Objects). In the MultiON 2.0 dataset with NAT-objects, we used the same episodes of the CYL-object one and only changed the objects placed in the environments.

Procedural Generation of Episodes. Compared to MultiON [176] based on only 90 scenes, MultiON 2.0 has 10x more environment. This allows the training of agents on a more differentiated distribution of scenes. Particularly, we base the episodes for MultiON 2.0 on Habitat-Matterport 3D Research scenes (HM3D) [135], a large-scale repository of a thousand indoor scenes. Mainly, for both tracks (CYL-objects and NAT-objects, we selected 800 training, 30 validation, and 70 test scenes. A clear comparison between MultiON and MultiON 2.0 datasets is in Tab. 4.1. To be compatible and complementary to prior work in MultiON, the episodes are also generated by sampling random navigable points as start and goal locations, such that

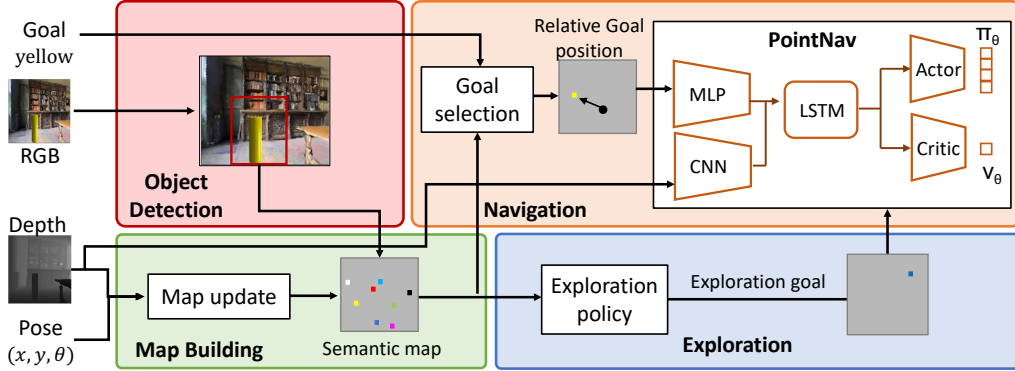


Figure 4.3: **Modular-MON**. We adopt a modular approach to multi-object navigation. The *Object detection* module transforms raw RGB to semantic labels. These are projected onto a top-down semantic map using depth observations by the *Map building* module. This serves as input for the *Exploration* module to uncover unseen areas of the environment. Finally, using exploratory or task goals, a low-level policy predicts the action for the agent to execute.

the locations are on the same floor and a navigable path exists between them. Next, we randomly sample n (corresponding to the number of goal objects, e.g., for 3ON we select three objects) objects from the set of NAT-objects (or cylinders, for the CYL-objects setup) to insert between the start and the goal. Additionally, we insert m distractor objects to make the task more challenging². The presence of distractors enables embodied policies to learn discriminability of goal objects, making success of random stumbling onto objects even more rare. The minimum geodesic distance between each pair of objects is kept at 0.6m in the training split to avoid goals from being cluttered together. We increase this to 1.3m in the validation and test splits to add variability to the data distribution. In Fig. 4.2 are shown two boxplots that analyze the difficulty of the MultiON 2.0 episodes (on the validation set). Mainly, Fig. 4.2-(a) compares the average episode length between 3ON2.0, 5ON2.0 from MultiON 2.0, and 3ON [176], ObjectNav [19]. Despite smaller environments in HM3D compared to the ones in Matterport3D (on which 3ON and ObjectNav are based), 3ON2.0 has, on average, $\sim 3x$ longer episodes than ObjectNav and is even longer than 3ON ($\sim 26m$ vs. $\sim 23m$). 5ON2.0 has a sequence of five object goals and has the longest average episode length. Instead, in Fig. 4.2-(b), we measure the average distance between every pair

²We set $m = 8 - n$ to make shorter horizon tasks to have more distractors than longer horizon tasks.

of objectgoals. Thanks to the constraint imposed between objects, also in this case, 3ON2.0 has a greater distance than 3ON. Interestingly, 5ON2.0 has slightly closer objects due to the increased number of objectgoal in the environments. These plots confirm that the episodes in MultiON 2.0 are more difficult than those in 3ON since the average shortest path to find every target is longer, and the objects are placed far apart.

4.4 Modular-MON

As we introduce in Sec. 4.1, with Modular-MON, we take a modular approach to multi-object navigation by employing the following modules: (1) *Object detection*, (2) *Map building*, (3) *Exploration* and (4) *Navigation*. These modules are intuitively weaved together. Modular-MON observes the environment and builds a semantic map by projecting information about category labels of the objects (*i.e.* semantics) in the field of view. If the agent has not yet discovered the current goal at hand, it will continue to explore and build the semantic map. Once the current goal has been discovered, Modular-MON plans a path from its current location to the goal. This process is repeated until the the policy has found all the goals, at which point the episode terminates. We experiment with different exploration and navigation strategies to systematically investigate their contribution to the agent performance. Next, we deep dive into each of these modules.

Object detection. Operating in high-fidelity simulation allows us to query semantic labels of the objects from sensors in the Habitat simulator. With these oracle semantic values, we can better investigate navigation and exploration strategies as a noiseless semantic map is assumed. Importantly, we experiment with a learned object detection model for a head-on comparison, where such a privilege is not assumed. To this end, we finetune (pretrained on MS-COCO) a FasterRCNN [138] architecture to detect the cylinders. Subsequently, we adopt a k-nearest neighbors algorithm to distinguish between different categories which is quite effective (as seen repeatedly in vision and robotics [16, 172, 70, 108, 121]). Particularly, we sample the RGB value from the center of each bounding box and feed it for KNN. If the most frequent neighbor has a cardinality is above a percentage threshold α_{KNN} , the color label is determined. For the *NAT-objects*, instead, we trained the FasterRCNN for the eight possible objects available in the dataset.

Map building. A cumulative memory representation is key for long-horizon tasks like multi-object navigation. To build such a representation, we project

the semantic detections onto a top-down grid map of the environment. The geometric transformation is done leveraging the depth channel of RGBD observations. Concretely, we compute point cloud from depth observations and register them into an allocentric coordinate system using agent poses use the mapping method in line with prior work [36]. The cumulative top-down map, therefore, contains semantic labels of the target and as well as the distractor objects. An intuitive mapping of each grid cell corresponds to 0.2 meters of the physical environment showed good performance and within memory budget for neural transforms. This map is used by both the Exploration module to sample exploration goals and the Navigation module to plan a path from the agent location to the goal and generate actions.

Exploration. For any policy to train well a tradeoff of exploration/ exploitation is imperative. This is particularly crucial for long-horizon tasks, where the agent has to tackle ambiguity for large intervals where the current goal is yet to be discovered. We investigate several simple-yet-effective exploration strategies, based on success in prior works.

- *Random*: sample a random exploration goal around the current location and keep moving towards it until the exploration goal is reached.
- *Random w/ threshold*: similar to *Random* with the difference that the agent moves towards the sampled point for α_{exp} number of steps before sampling another point. This is especially effective when the exploration goal is not reachable.
- *Stubborn* [107]: select an exploration goal in one direction and keep moving towards it until it runs into obstacles. After an obstacle is hit, pick another direction and repeat the process.
- *Stubborn w/ threshold*: this is similar to *Stubborn* but we limit the number of steps as we do in *Random w/ threshold* using the threshold α_{exp} .

Importantly, when Stubborn exploration is shown to outperform the two most widely-used exploration strategies – Frontier Based Exploration [190] and a deep net exploration policy SemExp [36] (study in Luo et al. [107]). Building on these findings and since our primary objective is to benchmark the new MultiON 2.0, we defer these additional exploration baselines for future work. Fig. 4.4 shows how the different methods select exploration goals.

Navigation. Given a location, either the exploration goal (output of Exploration module) or the object goal (location of the current goal on the

semantic map), this module plans a path towards. Particularly, it generate a sequence of actions to reach the location in a closed loop.

We formulate this a PointNav task and we employ a neural policy architecture. Concretely, it includes a visual encoder in the form of a ResNet50 backbone [76] (for depth observations), a multi-layer perceptron to transform GPS+Compass coordinated to latent representations, and a LSTM module [80] to capture state features from previous time steps. The latent representations from all these are concatenated to and transformed using two fully-connected layers *i.e.* the actor and policy heads to give the policy logits for the policy and state’s value function, respectively. The low-level actions for interaction with the environment are sampled from the predicted policy logits.

Beyond the parameterized neural PointNav policy, we also test an analytical path-planner. *BFS Path Planner* is an analytical path-planning algorithm, where the agent plans a path based on the 2D occupancy map from its current location to the goal location. The agent iteratively searches all its adjacent grid locations following a breadth-first search until it finds a path to the goal. The occupancy map of the environment is built following a similar mapping method as in our Map building module. The agent also builds a collision map by marking the grid cells where it collides. Furthermore we dilate the obstacles to keep the agent as far away as possible in order to avoid it getting stuck to corners and crevices. This is analogous to the pessimistic obstacle map from [107]. Also, if the agent fails to plan a path, we randomly sample one of the three actions: {move forward, turn left, turn right} to dislodge itself of a stuck position.

Implementation Details. The object detection model extracts the proposed bounding boxes from the input RGB, with a score threshold of 0.95. We set K in the KNN algorithm to 10. We set α_{KNN} to 80% *i.e.* if 8 of the 10 nearest neighbors is of the same category, we select that as the label. We find that a the step threshold α_{exp} of 50 works well for exploration. The same threshold is used for *Stubborn w/ threshold*. For *Stubborn*, following Luo et al. [107], we assume a local grid around the agent and select four corners of the grid in a clockwise direction as the exploration goals. For BFS Path Planner, we additionally provide corrections to the collision map to account for ‘invisible obstacles’ as a result of incomplete navigation meshes and inaccurate depth observations from the simulator.

Navigation Module		Validation				Test			
		Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
OracleReveal	PointNav [135]	84	90	37	41	81	85	36	39
OracleSem	PointNav [135]	85	89	39	40	81	87	37	39
	BFS Path Planner [58]	27	41	19	29	21	44	12	22

Table 4.2: **Quantitative results (navigation policies)**. We find that the analytical BFS Path Planner path-planner performs considerably worse than the learned PointNav on the 3ON (cylinders) task. We employ the best-performing Random w/ threshold across these baselines.

Exploration Module		Validation				Test			
		Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
OracleSem	Random	78	84	35	37	72	80	33	36
	Random w/ threshold	85	89	39	40	81	87	37	39
	Stubborn [107]	69	77	25	27	66	75	23	26
	Stubborn w/ threshold	75	82	35	38	72	80	33	36

Table 4.3: **Quantitative results (exploration strategies)**. We find a random exploration strategy with a step threshold performs better than the learned Stubborn [107] exploration strategy (and a step variant). The above experiments are on 3ON (cylinders) task. For all row entries, we employ the best-performing PointNav as Navigation module.

4.5 Experiments

We organize this section as follows. First, we include the necessary details about the multi-object navigation task. Second, we explain the embodied setup including sensor details for reproducibility. Third, we describe the standard metrics adopted for evaluating performance of long-horizon navigation tasks such as multi-object navigation. Fourth, we detail the mapping methods that we benchmark for multi-object navigation. Fifth, we include the quantitative results for the baselines. Finally, we include qualitative results and analysis.

4.5.1 Multi-Object Navigation Task

In MultiON [176] the agent needs to find and navigate to a fixed sequence of N objects in an unexplored environment. Specifically, the agent has access to (256×256) egocentric RGB image and depth map of the current view, current agent coordinates relative to its starting point in the episode through a noiseless GPS+Compass sensor, and the current goal category at a given time step of the episode. The agent can take one of four actions: *move forward*

	Validation				Test			
	Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
OracleSem	85	89	39	40	81	87	37	39
PredictedSem	31	47	17	19	27	45	15	17

Table 4.4: **Quantitative results (OracleSem & PredictedSem)**. The PredictedSem agent performs significantly worse than the OracleSem agent, due to error in the Object Detector. The above set of experiments are on 3ON (cylinders) folds using the best-performing modules for Navigation (PointNav) and Exploration (Random w/ threshold).

by 25 cm, *turn left* by 30° , *turn right* by 30° , and *found*. The agent has a maximum time horizon of 2500 steps to complete the task. Note that this is longer than standard navigation tasks due to the long-horizon nature of multi-object navigation. The agent must execute the *found* action within 1 meter of each goal for each of the N goals, in the right order, to be successful on this episode. A single incorrect *found* action terminates the episode – making the task very challenging. We use the widely adopted Habitat platform [145] for our experiments. The agent embodiment is a cylindrical body of 1 meter radius and 1.5 meter height. We conduct all experiments using the 3ON cylinders dataset containing 3 goal objects and five distractor objects.

4.5.2 Metrics

In addition to the standard visual navigation metrics such as *success* (whether the agent can reach all the targets successfully in the given sequence) and *SPL* [6] (Success weighted by inverse Path Length) we use the metrics introduced for the specifically for long-horizon tasks [176] such as multi-object navigation. These include *progress* (proportion of objects correctly found in the episode) and *PPL* (progress weighted by path length). The SPL and PPL metrics quantify efficiency of navigation in context of success/progress and increase if the agent trajectory better matches the optimal trajectory.

4.5.3 Baselines

We use a neural PointNav policy trained using the established distributed PPO [180] framework for efficient parallelization on HM3D scenes.

A key objective is to investigate the contribution of the different exploration and navigation approaches using the following mapping baselines.

The *OracleReveal* agent has access to the top-down oracle map of the environment directly obtained from the Habitat simulator marked with objects

(targets and distractors) observed by the agent during exploration. The ground-truth object locations are directly transformed to grid coordinates to build this map. Objects once seen remain seen for the length of the episode.

Using egocentric depth observations, the *OracleSem* agent builds a semantic map of the environment. We get the semantic labels of the objects (targets and distractors) directly from the Habitat simulator. This agent does not have access to the ground-truth locations of the objects.

The *PredictedSem* agent also builds a top-down semantic map following the same mapping method in *OracleSem*, but the egocentric semantic labels are predicted using a pre-trained object detection model.

4.5.4 Results

In Tab. 4.2 we first compare the *OracleReveal* agent with the *OracleSem* and found that the performances are similar. This is intuitive since the semantic maps in the two methods are similar in essence. In *OracleReveal*, we obtain a progressively revealed oracle semantic map directly from the simulator, while in the *OracleSem*, we project the semantic labels on to a semantic map using depth images. Next we show a comparison of the different navigation choices for our *OracleSem* agent. We observe that the pretrained PointNav policy performs significantly better than the analytical path planner in both validation and test sets. The performance gain is 21% to 81% (a $4\times$ increase) in success and 44% to 87% (a $2\times$ increase) in progress on the test set. Similar gains are seen on the SPL and PPL metrics, 12% \rightarrow 37% and 22% \rightarrow 39%, respectively.

In Tab. 4.3, we compare the different exploration strategies for the *OracleSem* agent using PointNav as the navigation policy. We find that a simple random policy with a step threshold outperforms the other exploration methods. Generally, we observe that the Random variants perform better than the Stubborn variants. This can be attributed to the fact that Random agents tend to explore more of the environment than Stubborn (Fig. 4.4), which tends to explore in smaller local areas. Specifically, in scenes with pockets of navigable areas connected by narrow corridors, Stubborn agents can tend to get stuck in a single area, whereas Random agents can cover more space.

We also compare the *OracleSem* agent with the *PredictedSem* in Tab. 4.4 and observe a drop in performance. This performance drop can be attributed to the error in the object detection module in *PredictedSem*. We report these numbers based on our experiments with the cylinder objects dataset. For results on natural objects, please refer to the supplemental.

	Trained Module	Trained On	Evaluated On	3ON Test set			
				Success	Progress	SPL	PPL
OracleSem	PointNav	HM3D	HM3D	81	87	37	39
			MP3D	21	37	11	20
No-Map [176]	end-to-end	MP3D	HM3D	0.4	6	0.2	3
			HM3D w/o dist.	1	13	0.5	6
			MP3D	10	24	4	14
ObjRecogMap [176]	end-to-end	MP3D	HM3D	0.3	10	0.1	0.3
			HM3D w/o dist.	3	18	0.8	6
			MP3D	22	40	17	30
ProjNeuralMap [176]	end-to-end	MP3D	HM3D	0.5	9	0.2	4
			HM3D w/o dist.	4	19	1	8
			MP3D	27	46	18	31

Table 4.5: **Transferability.** Above experiments show that the Modular-MON performs better than end-to-end models when transferred (without finetuning) to unseen environments. We observe that the OracleSem agent achieves 21% success on the MP3D scenes even when the PointNav, used as a Navigation module, is trained on HM3D scenes. On the other hand, none of the end-to-end models, trained on MP3D scenes transfers well to HM3D scenes.

4.5.5 Transferability of Modular-MON

To investigate the transferability to unseen environments, we evaluate Modular-MON on the MultiON dataset introduced by Wani et al. [176], based on Matterport 3D[32] scenes. We evaluate our OracleSem agent on the 3ON test set from MultiON. Tab. 4.5 shows that the OracleSem method achieves 21% success and 37% progress on the MP3D scenes even when the PointNav is trained on HM3D scenes. We found that this gap in performance is mostly due to the agent getting stuck on floating debris and holes on the floors in MP3D scenes. On the other hand, the end-to-end models do not transfer well from MP3D to HM3D. More specifically, we evaluate three models No-Map, ObjRecogMap and ProjNeuralMap from Wani et al. [176] on the MultiON 2.0 3ON test set based on HM3D scenes. The No-Map does not have access to any map, while the ObjRecogMap agent builds a map with predicted semantic categories of objects it sees. The ProjNeuralMap model, on the other hand, builds a global top-down environment map containing neurally projected image features. These agents are trained end-to-end with PPO[147] using a reward for 40 million frames and directly predict an action. We found that none of the models perform well when transferred to the HM3D scenes. The best performing model from [176], ProjNeuralMap, achieves 4% success and 19% progress on the HM3D test set without distractors and 0.5% success

	Validation				Test			
	Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
1ON	96	96	36	36	95	95	35	35
3ON	85	89	39	40	81	87	37	39
5ON	68	78	33	36	66	76	32	36

Table 4.6: **OracleSem performance on 1ON vs 3ON vs 5ON.** We observe that the agent performance deteriorates with more number of target objects.

and 9% progress with distractors. These experiments confirm that a modular approach transfers to unseen environments better than end-to-end approaches in the MultiON task.

4.5.6 Generalization of Modular-MON on n -ON

The modular approach of Modular-MON allows us to use the same modules for any n -ON tasks without retraining. This is more efficient and generalizable compared to end-to-end approaches[176] that need to be retrained every time we introduce more objects.

To study this, we evaluate the OracleSem agent on 1ON, 3ON, and 5ON episodes from both the validation and test sets. Although the performance decreases as we introduce more target objects for the agent to find (Tab. 4.6), with 1ON being the best and 5ON being the worst, the agent still performs considerably well across all n -ONs. The agent achieves a progress of 95% on 1ON, 87% on 3ON, and 76% on 5ON test set. Furthermore, the progress values on 3ON and 5ON are comparable to 86% ($95\% \times 95\% \times 95\%$) and 77% ($95\% \times 95\% \times 95\% \times 95\% \times 95\%$), respectively, if we were to treat each of the goals in 3ON and 5ON independently and reset the agent after it finds each goal. However, following similar calculations, we can see that the success on 3ON (81%) and 5ON (66%) are lower than 86% and 77% respectively. This can be explained by the fact that we keep the maximum number of steps fixed at 2500 for 1ON, 3ON, and 5ON, and the task gets even more complex for the agent in that it needs to find more objects within the same number of steps.

	Validation				Test			
	Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
CYL	31	47	17	19	27	45	15	17
NAT	25	37	14	12	20	30	10	17

Table 4.7: **PredictedSem performance on CYL-objects vs NAT-objects** The PredictedSem agent with Nat-objects performs worse than the one with the CYL-objects. This shows that natural objects with varying sizes, shapes and colors are harder to detect than cylinder objects with same sizes and shapes with varying colors. The above set of experiments are on 3ON folds using PointNav as the Navigation and Random w/ threshold as the Exploration module.

4.5.7 Object Detection on Natural objects

We introduce a set of *Natural-objects* (NAT-Objects) as part of our large-scale MultiON 2.0 dataset to increase the task complexity in terms of visual understanding. The eight natural objects (Fig. 4.5) are selected in a way that they are of varying sizes (toy train vs electric piano), shapes (guitar vs. basketball), colors (backpack vs. trolley bag), and at the same time can be found lying around inside houses. The objective was to transition from distinctive-looking cylindrical objects (CYL-objects) to more realistic-looking objects to make the task more interesting and challenging.

We perform experiments with the NAT-objects to study their differences in visual understanding with the CYL-objects. To this end, we finetune a FasterRCNN [138], pretrained on MS-COCO, to detect these eight possible natural objects. Unlike object detection on cylinders, we do not use an additional KNN to distinguish between colors. Instead, we expect the FasterRCNN to learn the different shapes, sizes, and colors of natural objects jointly. Tab. 4.7 shows a comparison of the PredictedSem agent with CYL-objects and Nat-objects. The performance on Nat-objects is worse than on CYL-objects, showing that natural objects with varying sizes, shapes, and colors are harder to detect than cylinder objects with the same sizes and shapes with varying colors.

4.5.8 MultiON 2.0 distractors vs. no distractors

Our MultiON 2.0 dataset additionally contains distractor objects in both CYL and NAT-objects episodes to make the episodes more challenging in terms of distinguishing between the goals and the distractors.

Distr.	Validation				Test			
	Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
X	86	89	41	42	81	90	37	40
✓	85	89	39	40	81	87	37	39

Table 4.8: **Effect of distractors on OracleSem performance.** We observe that the Modular-MON agent performs equally well in the presence of distractors. This can be attributed to our target location retrieval method from the semantic map comparing directly with the next goal category.

We thus perform experiments to study the effect of having distractors for our Modular-MON. We evaluate our OracleSem agent on both validation and test sets for 3ON with and without distractors. Tab. 4.8 shows that the Modular-MON performs equally well in the presence of distractors. This is intuitive since we select the target location on the global map containing semantic categories of both the targets and the distractors by directly comparing with the next goal category given as input to the agent.

4.5.9 Qualitative Results and Analysis

Fig. 4.6 shows an rollout of the OracleSem policy with PointNav and Random w/ threshold. During the first phase of the rollout, we can see that the agent keeps exploring the environment since it has not yet discovered the first goal. Once the agent has found and navigated to every goal, the episode terminates successfully.

Next, we compare the modules for navigation. Analytical path planners are computationally expensive and need handcrafted rules as compared to PointNav policy. While the PointNav policy takes in input depth observations and learns an internal representation of an obstacle map, analytical path planners require an obstacle map built from the depth observations. This is backed by our time runs, where found a full validation set run took 12 hours for PointNav but 48 hours for the BFS Path Planner.

We also find that obstacle map are left incomplete due to inaccurate depth information or holes in the mesh in the Habitat simulator. To mitigate this, the analytical path planning methods like BFS Path Planner require an additional collision map to mark where the agent has collided with obstacles.

Moreover, we analyzed preliminary error modes to infer that agents when get too close to an obstacle, they get stuck. This entails the need for a more pessimistic collision map which pads an obstacle to keep agents far away.

However, if the padding is too large, the agent fails to plan a path efficiently as it eats into the navigable space. Additionally, prior works [107] and us had to employ explicit handcrafted rules to get an analytical planner dislodged if it gets stuck for several steps in the same location. This makes analytical path planners a less desirable choice in navigation tasks compared to neural policies.

4.5.10 Qualitative Examples

Fig. 4.7 shows a rollout of the OracleSem agent on one of the episodes from the 5ON test set. At each step the agent receives the egocentric depth and semantic observations along with the current goal category as inputs (column 1) and builds a top-down semantic map (column 3) from the egocentric object categories that it observes using the depth image. The agent switches between the Exploration and Navigation modes depending on whether it has seen the current target object. From the example, we see that the agent mostly explores the environment in the initial phase of the rollout. Once it starts discovering target objects, it navigates to them sequentially. Once it is able to successfully find all 5 objects, the episode terminates.

Fig. 4.8 and Fig. 4.9 show rollouts of the PredictedSem agent on the 3ON test set episodes with CYL and NAT objects respectively. Here the agent has access to the RGB and depth observations and the current goal category as inputs (column 1). The agent predicts the egocentric semantic category of the objects from the RGB image (column 2 shows the bounding box for the predicted object) and progressively builds a top-down semantic map (column 4) with the object categories using depth image. These examples also demonstrate that the agent mostly explores the environment in the first phase of the episodes, later switching to the Navigation mode once it discovers the target objects.

Finally, we look at some of the frequently occurring failure cases in Fig. 4.10. We found that it is sometimes possible for the agent to keep exploring one area in the house more than the other, leading to some goals not being discovered. We also found that the PointNav agent, used as our Navigation module, often fails to generate the *found/stop* action even when the agent has discovered the goal. The agent may also run out of the maximum steps quota (2500 steps) before discovering and navigating to all the goals in the episode. Additionally the agent may *stop* at a distance too far away i.e. more than 1 meter from the goal, which immediately terminates the episode leading to a failure.

4.6 Conclusion

We carried out a systematic analysis of a modular approach, Modular-MON, to the multi-object navigation (MultiON) task. To do this, we created a new large-scale dataset with both synthetic and natural objects placed in HM3D environments. Using this dataset, we compare various strategies for navigation and exploration. Our experiments show that deploying a PointGoal navigation agent in the MultiON task significantly outperforms analytical path planning. Moreover, a simple random exploration strategy surprisingly outperforms more complex heuristics. We believe our work offers insight for more efficient, modular approaches to MultiON in future work and encourages the Embodied AI community to explore a hybrid combination of transfer learning and simple heuristic-based methods.



Figure 4.4: **Visualizing exploration strategies.** (*Top*) shows a *Stubborn* strategy which selects one of four corners of a local map in a clockwise direction. (*Bottom*) shows a *Random* strategy where the agent randomly samples exploration goals. We also employ corresponding variants, particularly, *Random w/ threshold* and *Stubborn w/ threshold* that have a step threshold (see Sec. 4.4).

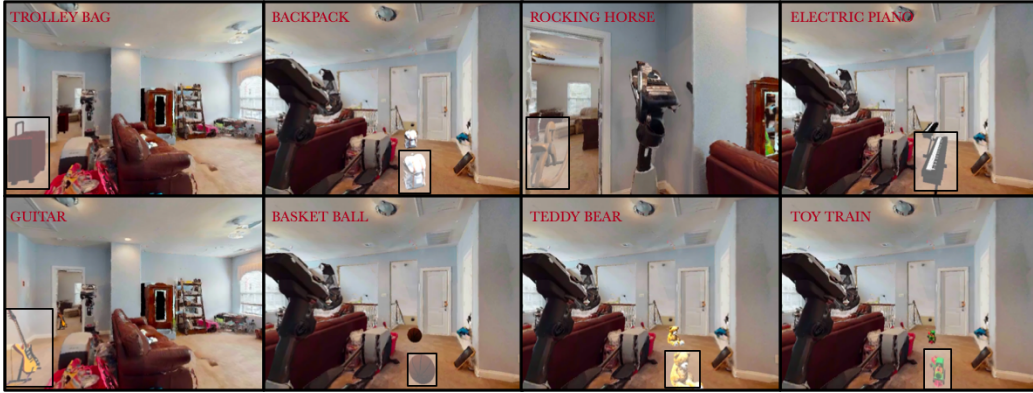


Figure 4.5: **Natural objects.** The set of eight NAT-objects vary in shapes, sizes and colors and easily blend in the HM3D houses, thus requiring better visual understanding for the agent.

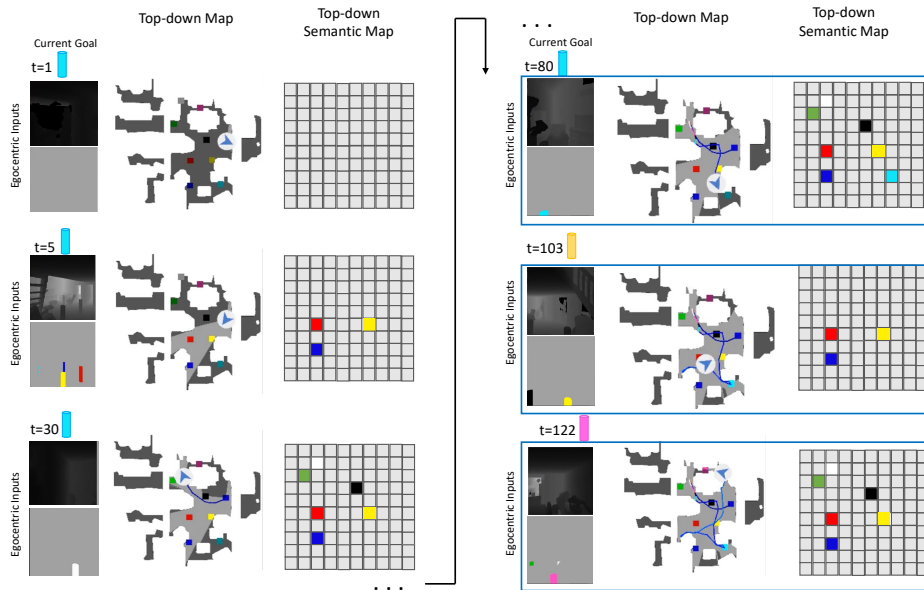


Figure 4.6: **Qualitative results.** Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and discovers objects and progressively builds the semantic map using egocentric depth observations. The goal sequence is (cyan, yellow, and finally pink). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the *found* action.

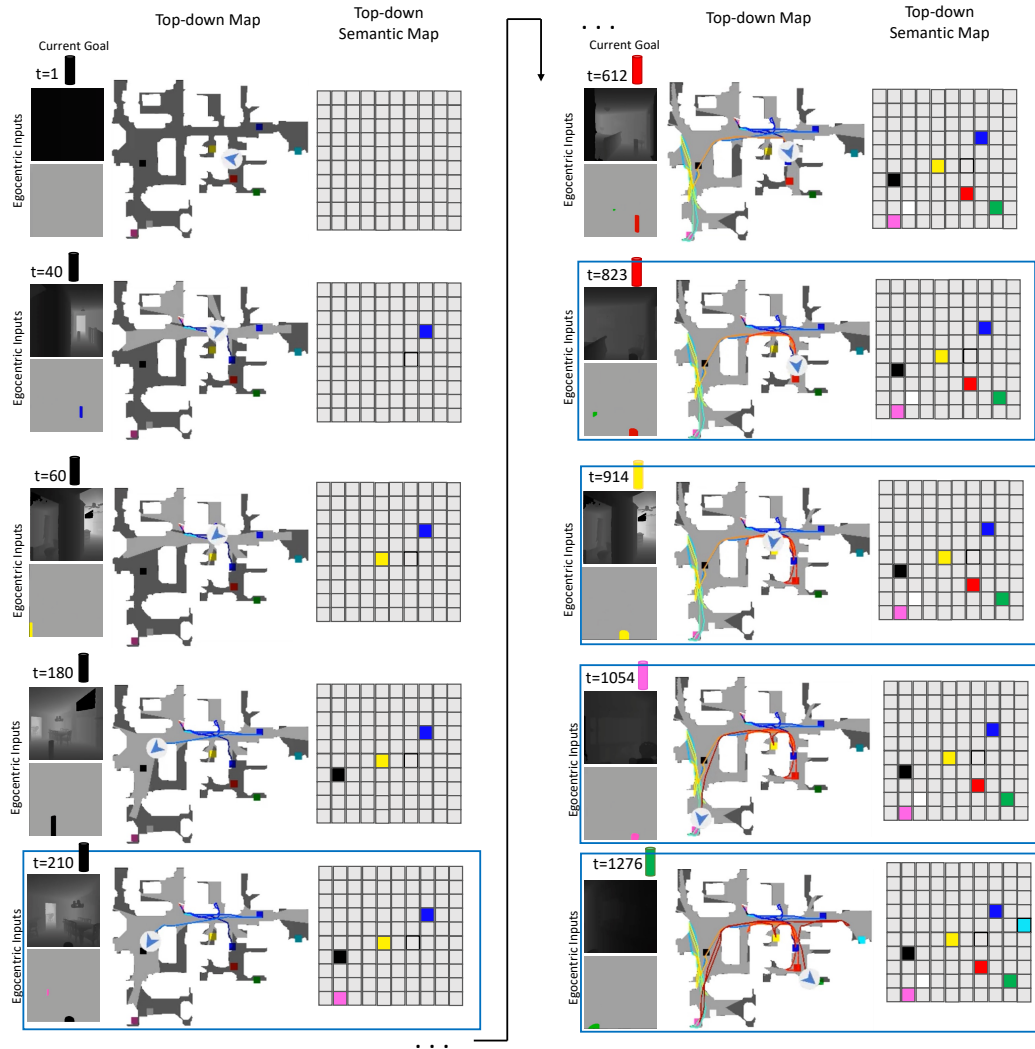


Figure 4.7: **Qualitative results: 5ON.** Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and discovers objects and progressively builds the semantic map using egocentric depth observations. The goal sequence is (black, red, yellow, pink, and finally green,). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the *found* action. The agent has a 100% Success, 100% Progress, 39% SPL and 39% PPL in this episode.

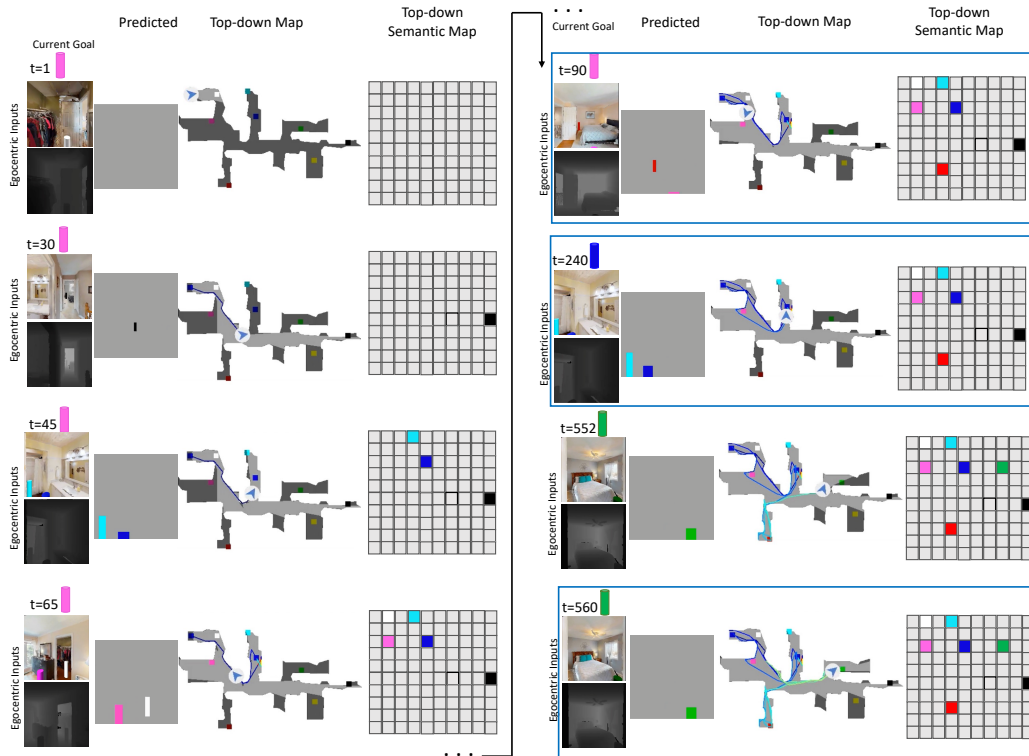


Figure 4.8: **Qualitative results: CYL objects.** Rollouts of our OracleSem with PointNav and Random w/ threshold show that the agent explores over time (t) and detects objects (‘Predicted’ column) and progressively builds the semantic map using egocentric depth observations. The goal sequence is (pink, blue, and finally green). The top-down obstacle map is for visualization only; this agent does not have access to it. Blue outline indicates that the agent executed the *found* action. The agent has a 100% Success, 100% Progress, 21% SPL and 21% PPL in this episode.

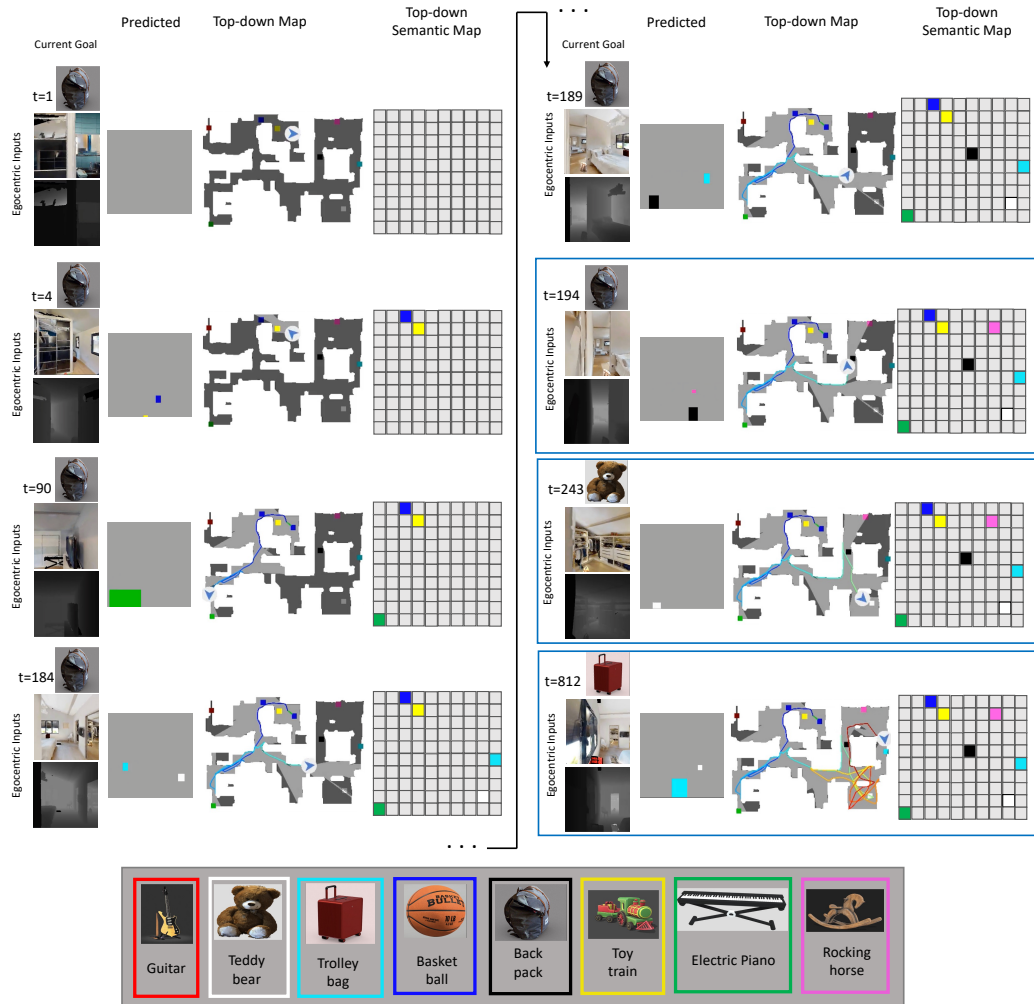


Figure 4.9: **Qualitative results: Natural objects.** Rollouts of our Oracle-Sem with PointNav and Random w/ threshold show that the agent explores over time (t) and discovers target objects and progressively builds the semantic map using egocentric depth observations. The goal sequence is (backpack (black), teddy bear (white), and finally trolleybag (cyan)). The top-down obstacle map is for visualization only; this agent does not have access to it. **Blue** outline indicates that the agent executed the *found* action. The agent has a 100% Success, 100% Progress, 17% SPL and 17% PPL in this episode.

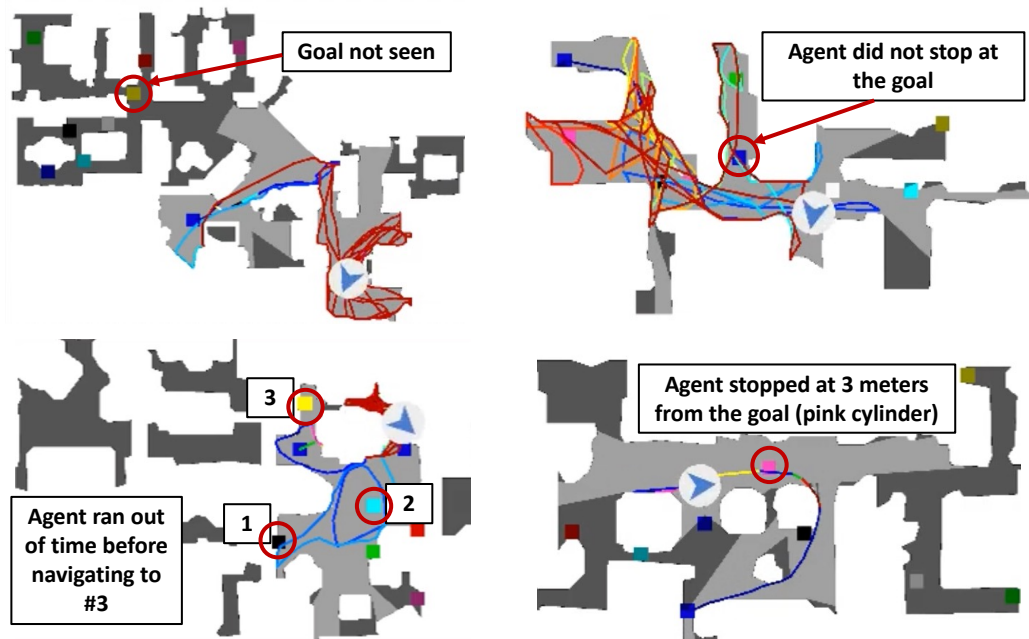


Figure 4.10: **Failure cases.** Most frequent failure cases include (clockwise from top left) (1) Goal has not been discovered before reaching maximum steps quota (2500); (2) PointNav fails to generate *stop* action; (3) Agent stops too far from the goal (>1 meter), leading to termination of the episode; (4) All goals have been discovered but maximum steps quota has been reached before navigating to the later goals.

Chapter 5

Exploiting Socially-Aware Tasks for Embodied Social Navigation

In this Chapter, we aim to provide a solution for **R1** and **R3**, detailed in Chapter 1. We are particularly interested in understanding how an agent can navigate cluttered environments populated by humans. Thus it is fundamental to avoid every kind of collision with pedestrians. To this end, we developed an end-to-end approach [29], where the agent learns the concept of *immediate* and *future* danger through the use of specifically designed Socially-aware auxiliary tasks. The notion of danger was directly derived from common-sense knowledge (i.e., humans, when facing others, slow down and try to avoid every possible collision). In addition, we define a fine-grained evaluation setup to directly evaluate different interactions between humans and the agent. We tested our approach in the Social Navigation task, where the agent must reach a specific coordinate in the environment by avoiding collision with humans.

5.1 Introduction

¹Navigating safely in a dynamic scenario populated by humans who are moving in the same environment is necessary for embodied agents such as home assistants robots. To do so, as depicted in Figure 5.1, the agent should be able to dynamically and interactively navigate the environment by avoiding static objects and moving persons.

Recently, the development of photorealistic 3D simulators [145, 153, 90] has provided the tools to train embodied agents and experiment in large-

¹E. Cancelli*, **T. Campari***, L. Serafini, A. X. Chang, L. Ballan. "Exploiting Socially-Aware Tasks for Embodied Social Navigation". Under submission. – * denotes equal contribution.

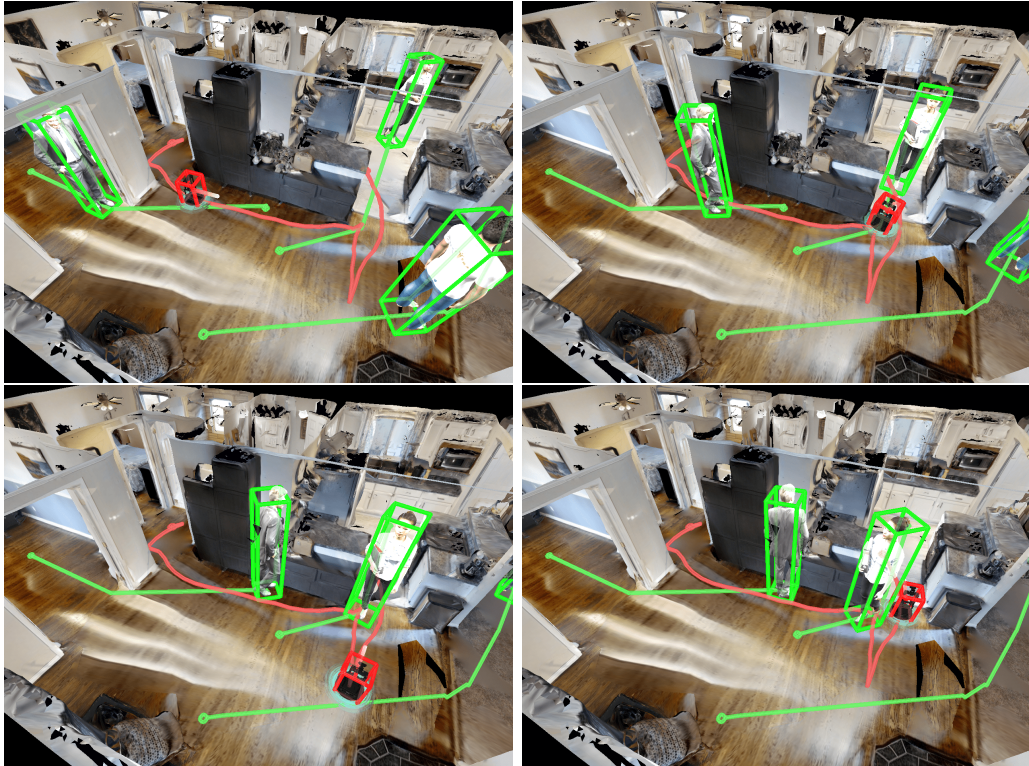


Figure 5.1: Illustration of an agent-person “encounter”. From top-left to bottom-right: *i* episode starts; *ii* the embodied agent/robot sees a person; *iii* it moves back to avoid a collision; *iv* it successfully reaches the goal by avoiding the person.

scale indoor environments [32, 133, 59]. Thanks to these frameworks, several tasks, and challenges have been presented [6, 200, 58]. In particular, in the PointGoal Navigation task (where an agent is required to reach a specific location in an environment), an agent without any sensor/actuation noise trained for billions of steps can obtain almost perfect performance [180]. Other approaches [194, 123] obtained impressive results even in the presence of noise. Another relevant task is Object Goal Navigation, where an agent is required to find and navigate to a specific object instance. This task requires both semantic and navigation capabilities; to this end, modular approaches based on semantic-maps [36, 27, 136], as well as end-to-end models [194, 133] have been presented lately. High-level semantic understanding is even more critical in Vision-Language Navigation [174, 141, 199].

However, although challenging and despite encouraging progress, all the previously mentioned tasks frame navigation in a fundamentally static environment. The dynamic element introduced by sentient, moving human beings

in the scene forces us to rethink how the current models are designed. A good navigation policy must not be just effective (i.e., able to achieve its goal) and efficient (i.e., able to achieve the objective through a close-to-optimal path) but also safe (reaching the destination without harming others). This social element is included in the Social Navigation Task [186, 128], where an agent must tackle PointGoal Navigation in simulated, indoor, and crowded environments. In this scenario, [196] recently introduced a simple but quite effective model, although the approach can not explicitly encode any social behavior in its navigation policy. We believe that a clear encoding of human-agent interactions, as well as social behaviors, are required in complex scenarios in which the embodied agent cooperates and interacts with humans. In this way, the agent could prevent collisions or dangerous behaviors and adapt its path to the dynamic environment in which it is navigating. We encode these “signals” by introducing two *Socially-Aware Tasks*, referred as *risk* and *social compass*. These tasks model the present and future danger connected to the agent’s action.

Additionally, we define an extensive evaluation protocol for the Embodied Social Navigation task in order to better analyse the performances in case of human-agent interactions. This is inspired by a similar attempt that has been recently introduced in robotics [131], consisting in collecting statistics about specific encounters between humans and a robot (through questionnaires). We propose an automated procedure for fine-grained human-agent interactions. To this end, given a specific episode, we extract short sub-sequences of interest in which a social interaction becomes the predominant factor influencing navigation, called *encounters*. Each encounter is associated with a corresponding category based on the type of human-agent interaction occurring, following a set of predetermined rules. Finally, we also created a dataset for Embodied Social Navigation to assess our agents in different environments. This dataset was built on top of HM3D [133].

In summary, the contributions of this work are threefold:

- A novel architecture for embodied social navigation which is based on Socially-Aware tasks; we prove the effectiveness of the model on two public datasets.
- A new Encounter-based social evaluation protocol.
- An extended dataset for Embodied Social Navigation based on the popular HM3D dataset (called HM3D-S).

Socially-Aware Navigation. Socially/Human Aware representations and models have been studied by several researchers in the field of robotics,

computer vision and human social behavior analysis [117]. Some works focused on collision avoidance algorithms that, similarly to earlier models like ORCA [21] or RVO [168], try to enable collaborative, collision-free navigation of non-communicating agents in a shared space. Modern approaches employ Deep Reinforcement Learning (DeepRL) to learn a motion policy that can produce a safe path to a goal for every agent, by enabling efficient online prediction of future states of its neighbours [47, 105].

Other works explicitly tackle the problem of motion planning and navigation in environments with dynamic obstacles [9] and/or humans [63, 46, 106, 40]. [46] employs collision avoidance algorithms like CADRL [47] and introduces common-sense social rules to reduce uncertainty while minimizing the risk of incurring in the Freezing Robot Problem [166]. [63, 40], instead, try to model human-agent interaction using techniques like Spatio-temporal graph [106]. These methods have been widely tested on minimalist simulation environments that provide complete knowledge and simple obstacles and often assume collaboration between moving agents. Limited tests have been conducted on real-world scenarios, but they often require a large set of sensors for free-space detection, mapping obstacles, and sensing humans [46].

Recently, an increasing amount of works have been focusing on exploiting egocentric visual data to learn navigation policy to operate in social environments with partial information. [163] used depth maps to train an agent using imitation learning and studied the policy behavior in a set of simulated interactions. [128], instead, focuses on constrained indoor environments and uses a combination of a global planner with complete map access and a low-level RL policy that exploits data from a LIDAR. This approach was tested on a set of simulations and evaluated using both standard metrics and domain-specific like *Human Collision Rate*. However, this approach requires prior knowledge about the environment where it operates for the path planner to work.

Embodied Navigation. Embodied Navigation had a surge in the last years [58]. Mainly, this was possible thanks to large-scale datasets consisting of 3D indoor environments [32, 153, 133], and to simulators that allow researchers to simulate navigation inside these 3D environments [145, 153, 90]. In this context were proposed many tasks [6] such as: PointGoal Navigation [180], ObjectGoal Navigation [19], Embodied Question Answering [179], and Vision and Language Navigation (VLN) [7, 93]. To tackle these problems, where an agent operates in static, single-agent environments, modular approaches were proposed [28, 37, 36, 136], exploiting SLAM, Path Planning and exploration strategies, and end-to-end RL-trained policies [176, 123, 195, 27, 194], without

exploiting any explicit map. In this chapter, we will focus on the Social Navigation Task [186, 128], where the agent has to navigate in environments with moving humans. This adds new challenges to Visual Navigation since social capabilities are required to avoid collisions. Modular approaches are harder to adapt in this context since humans are constantly moving and therefore harder to track.

In [196], Yokoyama et al. proposed an end-to-end RL-trained policy for Social Navigation. This model extracts embeddings from the Depth and the GPS+Compass sensors and feeds them to a GRU, together with the previous action. However, this model did not exploit any Social information and was evaluated only using success rate, which is limiting since the agent is dealing with humans and it is preferable to navigate safely in order to avoid collisions, even if this means having lower success rate. There have been some attempts to do a fine-grained evaluation for Social Navigation. For example, [131] defined an evaluation protocol for social agents based on human questionnaires. This cannot be easily applied to simulations, where test sets contain thousands of episodes. In this chapter, we will propose an automatic evaluation protocol to measure the social capabilities of our models.

5.2 An evaluation protocol for SocialNav

SocialNavigation Task. In *Social Navigation* [186, 128, 196], as in Point-Goal Navigation, the agent aims to reach a target location, but a collision with a human subject constitutes a failure and will terminate the episode. An episode e is characterized by the agent trajectory α , the tuple of human trajectories (\mathbf{p}^i) , and the target goal g .

The agent trajectory α is a sequence of positions and rotations of the agent from the beginning to the end of the episode t_{end} . Formally, $\alpha = \{\alpha_t\}_{t \in [0, t_{\text{end}}]}$ where $\alpha_t \in SE(2)^2$ is the 2D translation and rotation of the agent with respect to the origin at time t .

Similarly, the trajectories of humans in the episode are sequences of positions and rotations associated with the i -th human. Formally, $\mathbf{p}^i = \{p_t^i\}_{t \in [0, t_{\text{end}}]} \forall i \in \mathcal{P}$ with $p_t^i \in SE(2)$. In our simulation, the movement of each person is constrained by an associated starting point and an endpoint, with the person moving back and forth between those two points following the shortest path.

The target goal $g \in \mathcal{G}$ is specified by the 2D position in world coordinates.

²SE(2) is the 2-dimensional special euclidean group.

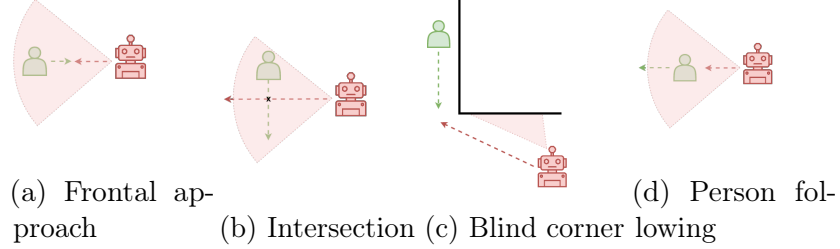


Figure 5.2: A scheme representing the four different classes of encounter. The dashed line represents the general direction of the agent and the person involved. The red area represents the agent’s field of view at the beginning of the encounter.

The agent must at any point in time provide an action $(\text{lin_vel}, \text{ang_vel}) \in [-1, +1]^2$, representing the normalized linear forward velocity and the normalized clockwise rotational angular velocity (where +1 is the maximum velocity and -1 the maximum backward/counter-clockwise velocity). The stop action is automatically called when the agent is within 0.2 meters from the target goal point. The agent has 500 actions (or steps) to reach the target location. If it collides with a human, the episode terminates immediately.

5.2.1 Evaluation Protocol

Given an episode e , we define an *encounter* as follows:

Definition 5.2.1. An *encounter* taking place in episode e between the agent and a specific pedestrian $i \in \mathcal{P}$, is defined as a subsequence of trajectories α and \mathbf{p}^i in a given timeframe $[t_1, t_2] \subseteq [0, t_{\text{end}}]$ such that the following conditions are met:

- **Time Constraint:** the timeframe $[t_1, t_2]$ is larger than a threshold T_{min} ;
- **Spatial Constraint:** the geodesic distance between the agent and person $i \forall t \in [t_1, t_2]$ is less than a threshold D_{max} ;
- **Heading Constraint:** person i is in front of the agent for the first T_{front} timesteps. That is, given the agent’s heading angle θ_t^a , $\theta_t^{a \rightarrow i}$ the angle of the segment connecting the agent to person i and Θ_{max} a threshold, $|\theta_t^a - \theta_t^{a \rightarrow i}| \leq \Theta_{\text{max}}$ holds $\forall t \in [t_1, t_1 + T_{\text{front}}]$.

Encounter classification. To distinguish the encounters between the agent and a human subject, we devise a heuristic called *inclusion rule (IR)*. The IR is defined according to the following parameters:

- Δ_t^i, Δ_t^a represent, respectively, an approximation of the general direction of the trajectory of the agent and a person i in the timeframe $[t_1, t]$ with $t \in [t_1, t_2]$, where t_1, t_2 are the start and the end timesteps of an encounter;
- `intersect` is a binary value that is 1 if robot and pedestrian paths intersect and 0 otherwise;
- `blind(t)` is a time-conditioned binary value indicating whether the agent can see the person at time step t ;
- `d_diff(t)`: difference between the geodesic and the euclidean distance between the agent and the person.

Subsequently, we defined four categories (inspired by [131]), and their respective *inclusion rule* (see Figure 5.2):

- **Frontal approach:** The robot and the human come from opposite directions and have trajectories that are roughly parallel. In this context, the agent must deviate slightly to avoid a frontal collision. IR: $(\neg \text{blind}(t) \forall t \in [t_1, t_1 + T_{\text{view}}]) \wedge \pi - \Delta_{\text{slack}} \leq |\Delta_{t_2}^i - \Delta_{t_2}^a| \leq \pi + \Delta_{\text{slack}}$, where Δ_{slack} is a slack value (in radians) on the angle π and T_{view} is the number of initial timesteps in which the person must be visible by the agent.
- **Intersection:** The robot and the human’s trajectory intersect at approximately 90° . In this situation, an agent may want to stop and yield to the human or decrease its linear velocity and slightly deviate. IR: $(\neg \text{blind}(t) \forall t \in [t_1, t_1 + T_{\text{view}}]) \wedge \frac{\pi}{2} - \Delta_{\text{slack}} \leq |\Delta_{t_2}^i - \Delta_{t_2}^a| \leq \frac{\pi}{2} + \Delta_{\text{slack}} \wedge \text{intersect}$.
- **Blind Corner:** An agent approaches a person from an initially occluded position, like a corner or a narrow doorway. In situations with limited visibility like this, the agent should act cautiously to avoid sudden crashes. IR: $(\text{blind}(t) \forall t \in [t_1, t_1 + T_{\text{blind}}]) \wedge \text{d_diff}(t_1) \leq 0.5$ where T_{blind} is the number of initial timesteps in which the person must not be visible by the agent.
- **Person following:** A person and the agent travel in the same direction. The agent must maintain a safe distance from the person and a relatively

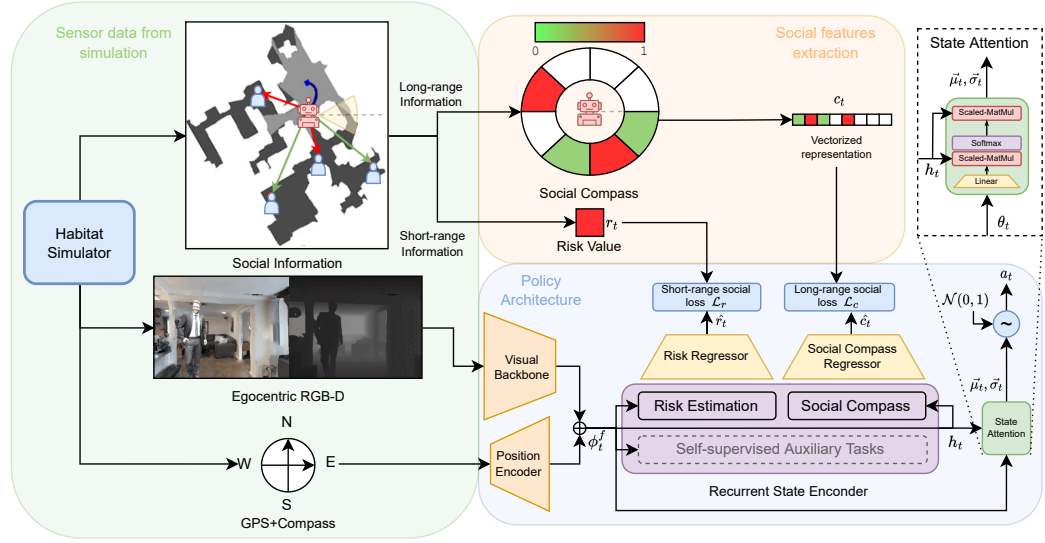


Figure 5.3: Pipeline and model overview. *Social information* is extracted from Habitat Simulator (left rectangle) and is processed through a *Social Feature extraction* procedure (top-right). The policy (bottom-right) uses RGB-D and GPS+Compass data as input and, during training, is conditioned by the extracted social features.

low linear velocity. IR: $(\neg \text{blind}(t) \forall t \in [t_1, t_1 + T_{\text{blind}}]) \wedge |\Delta_{t_2}^i - \Delta_{t_2}^a| \leq \Delta_{\text{slack}}$.

Metrics. For each encounter category, we compute the following metrics:

- **Encounter Survival Rate (ESR)** is the percentage of encounters (in a specific category) without a human collision (e.g., if in the Blind Corner encounter the agent collided with a human in the 20% of the cases, the ESR will be 80%);
- **Average Linear-Velocity (ALV)** is the average linear velocity of the agent in an encounter;
- **Average Distance (AD)** is the average distance of the agent w.r.t. the human in an encounter.

5.3 Method

Overview. Figure 5.3 shows an outline of our framework. It comprises two main modules: (i) *Social feature extraction*, and (ii) *Policy architecture*. The

Social feature extraction module refines social information obtained from the simulator to extract features that describe some aspect of social interactions (ground truth social features). The *Policy architecture* extracts from the RGB-D and the GPS+Compass sensors an embedding that serves as input for our Socially-Aware tasks. These tasks refine this embedding and create n embeddings (one per task). These embeddings are then fused together through state attention. From the state attention output is then sampled an action. In the following, we will detail the whole architecture.

5.3.1 Policy Architecture

Our policy network comprises the following modules: *i*) two encoders (the *Visual backbone* and the *Position Encoder*) that create an embedding from the RGB-D and the GPS+Compass sensors; *ii*) a *Recurrent State Encoder* that accumulates such embedding through a series of recurrent units; *iii*) a *State Attention* module that fuses the outputs of such units through an attention mechanism to produce the action the robot has to perform.

Each RGB-D frame x_t is encoded in a ϕ_t^v embedding using a CNN (*Visual Backbone*) $f(\cdot)$ such that $\phi_t^v = f(x_t)$. To encode the position and rotation of the agent α_t in a ϕ_t^p embedding, we used a linear layer $g(\cdot)$ such that $\phi_t^p = g(\alpha_t)$. Subsequently, the outputs of these two encoders are concatenated into the final embedding $\phi_t^f = \phi_t^v \oplus \phi_t^p$. To accumulate embeddings over time, we decided, similarly to what has been done in [195] for PointGoal Navigation, to implement our state encoder as a stack of parallel recurrent units. Each unit at each timestep is fed ϕ_t^f , and it outputs its internal state, called *belief*.

The key idea of having multiple beliefs is that each recurrent unit focuses on a specific navigation aspect. The final decision about what action the robot should take is sampled by weighting each belief according to the situation. For this reason, all beliefs \mathcal{B} are subsequently fused through the *State Attention* module to calculate the mean $\vec{\mu}_t$ and standard deviation $\vec{\sigma}_t$ of the normal distribution from which we sample the action a_t . More formally, given $\{RU^{(i)}\}_{\forall i \in \mathcal{B}}$ a set of recurrent units, the encoded beliefs h_t are defined as follows:

$$h_t := \{h_t^{(i)}\}_{\forall i \in \mathcal{B}} \leftarrow \{RU^{(i)}(h_{t-1}^{(i)}; \phi_t^f)\}_{\forall i \in \mathcal{B}} \quad (5.1)$$

The fusion mechanism of the state attention module SA is defined as:

$$\vec{\mu}_t, \vec{\sigma}_t \leftarrow SA(h_t, \phi_t^f) = FC_a(\text{Attention}(h_t, FC_k(\phi_t^f), h_t)) \quad (5.2)$$

where $\text{Attention}(Q, K, V) \mapsto \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V$ and FC_a and FC_k are two linear layers.

5.3.2 Socially-Aware Tasks

With multiple beliefs, we can inject different signals in our embeddings, e.g., social dynamics occurring in an episode. To this end, during training, we condition each belief with a unique auxiliary loss jointly optimized with the action and value ones during the optimization step of the policy network. This is done by processing each belief with a specific type of *Social feature*, through a *Regressor network* (see Fig. 5.4), that computes our *Socially-Aware tasks* predictions. Such tasks consist in the prediction of social features in $[t, t + k]$, conditioned by the corresponding belief $h_t^{(i)}$ and the sequence of performed actions $\{a_j\}_{j \in [t, t+k]}$, where k is the number of future frames to predict. Formally, for a given sequence of social features $\{s_j\}_{j \in [t, t+k]}$, the task aims to optimize the following auxiliary loss:

$$\mathcal{L}_f = \frac{\sum_{j \in [t, t+k]} \text{MSE}(s_j, \hat{s}_j)}{k} \quad (5.3)$$

where $\{\hat{s}_j\}_{j \in [t, t+k]} = \mathcal{M}(h_t^{(i)}, \{a_j\}_{j \in [t, t+k]})$ and \mathcal{M} is the regressor network. We designed two types of social tasks corresponding to two social features: (i) *Risk Estimation*, and (ii) *Social Compass*. Such design has the benefit of being easily extensible with other, possibly more complex social tasks and to be also compatible with general purpose self-supervised tasks like the ones used in [195] (e.g., CPC|A [72] or ID [125, 194]).

To exploit different social features, we extract from the simulator the relative position of every person w.r.t. the agent. We refer to this data as *Social Information*:

$$SI_t \stackrel{\text{def}}{=} \{\delta_t^i := (\text{pos}(p_t^i) - \text{pos}(\alpha_t)) \in \mathbb{R}^2\}_{\forall i \in \mathcal{P}}$$

where the function $\text{pos}(\cdot)$ extracts the position from an element of α or \mathbf{p}^i .

Risk Estimation. *Risk Estimation* is a Socially-Aware Task designed to deal with short-range social interactions, to inform the agent about imminent collision dangers. Given SI_t , we define the *Risk value* as a scalar representing how close the agent and the nearest person are up to a maximum distance D_r . This value ranges from 0 (the nearest neighbor is further than D_r meters away) to 1 (the agent and person are colliding). Formally:

$$\text{risk}_t = \text{clamp} \left(1 - \frac{\min\{\|\delta_t^i\|_2 \mid \delta_t^i \in SI_t\}}{D_r}, 0, 1 \right) \quad (5.4)$$

where $\text{clamp}(\cdot, 0, 1)$ limits the value to the $[0, 1]$ range.

Social Compass. Complementary to Risk Estimation, this *Socially-Aware Task* deals with the long-distance component of social dynamics. This feature captures not only social interaction on a larger area with radius $D_c > D_r$ but also a weak indication of the direction a person may come. Much like humans can make guesses about people’s whereabouts based on previous observations, partial knowledge of the environment topology, and a person’s trajectory; we expect to provide similar knowledge at training time while being easy to infer at evaluation time.

Such information is represented through a *Social Compass*. In the compass, north represents the direction the agent is looking, and the quadrant is partitioned into a finite number of non-overlapping sectors. Given each person $i \in \mathcal{P}$, $\theta_{a \rightarrow i}$ represents the angle of the segment connecting the agent to that person w.r.t. the north of the compass. These angles are associated with a specific sector. We compute the risk value for each sector among people in the same quadrant. Finally, we represent the compass in vectorial form by unrolling the sequence of sectors from the north going clockwise. Formally if we have k number of equivalent sectors, the vector $\text{comp}_t \in \mathbb{R}^k$ is defined as:

$$\begin{aligned} \text{comp}_t[j] &= \left[\text{clamp} \left(1 - \frac{\min\{\|\delta_t^i\|_2 \mid \delta_t^i \in \Theta_j\}}{D_c}, 0, 1 \right) \right] \\ \text{with } \Theta_j &= \left\{ \delta_t^i \in SI_t \mid \theta_{a \rightarrow i} \in \left[\frac{2\pi}{k} \cdot j, \frac{2\pi}{k} \cdot (j+1) \right) \right\} \\ &\quad \forall j \rightarrow [0, k-1] \end{aligned} \quad (5.5)$$

5.3.3 Implementation details

The visual backbone used in our experiments is a modified version of a ResNet-18 for RGB-D frames. We use the same implementation used in [196]; this produces an output embedding of size 512. For the agent’s position encoder, we use a linear layer with an output size of 32. Our recurrent state encoder is implemented as a stack of single-layer GRUs with a hidden dimension of 512 each. Regressors for both social tasks are implemented as single-layer GRUs with hidden size 512. The sequence of actions $\{a_i\}_{i \in [t, t+k]}$ is passed through an embedding layer and then fed to the input of the GRU. The initial hidden state is initialized to $h_t^{(i)}$, and k is set to 4. See Figure 5.4 for the complete scheme.

Each model was trained on 3 GPUs Nvidia A5000 with 8 parallel environments for each machine using the DD-PPO [180] algorithm. Our reward function is the following:

$$r_t = -\Delta_d + r_{\text{slack}} - \beta_{c \wedge b} \cdot (I_{\text{coll}} + \cdot I_{\text{back}}) + \beta_{\text{succ}} \cdot I_{\text{succ}} \quad (5.6)$$

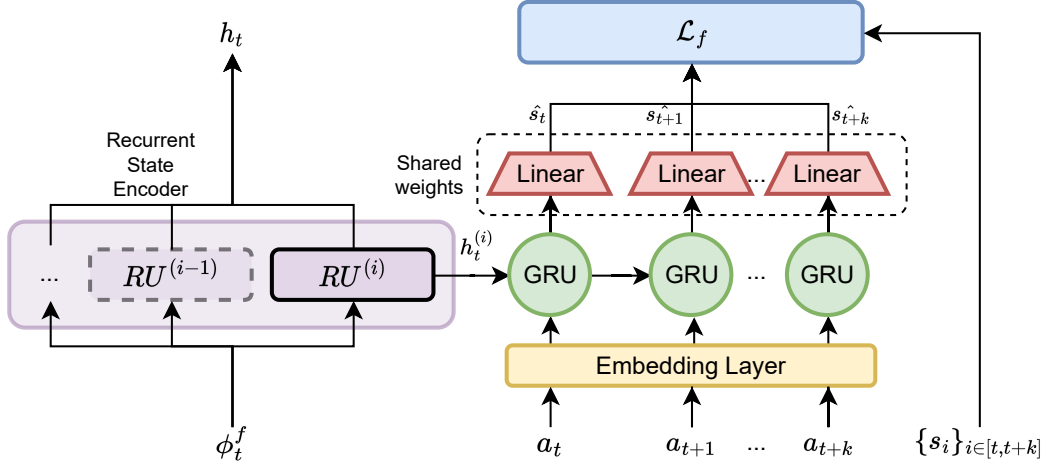


Figure 5.4: Implementation of a regressor network. Actions $a_t..a_{t+k}$ are used as input. A linear layer compresses the GRU’s hidden states to obtain the predicted social features $\hat{s}_t.. \hat{s}_{t+k}$. $\{s_i\}_{i \in [t, t+k]}$ is the ground truth used by \mathcal{L}_f (from Equation 5.3).

where Δ_d is the potential reward based on the geodesic distance to the goal, r_{slack} is the slack reward, and $I_{\text{coll}}, I_{\text{back}}, I_{\text{succ}}$ represent the indicator functions respectively of a collision with objects in the environment, the linear velocity being less than 0 and success. $\beta_{c \wedge b}$ and β_{succ} are coefficients. We used the same parameters as in [196], that are -0.002 for r_{slack} , 0.02 for $\beta_{c \wedge b}$ and 10.0 for β_{succ} .

5.4 Experiments

Name	Sensors		Aux Tasks			Social Tasks		Metrics (Gibson4+)			Metrics (HM3D-S)		
	RGB	Depth	CPCA	GID	CPCA/B	Risk	Compass	Success	SPL	H-Collisions	Success	SPL	H-Collisions
Baseline [196]		✓						72.65±1.6	47.43±1.2	24.35±1.9	62.76±2.2	36.69±1.1	29.29±2.2
Baseline+RGB [196]	✓	✓						74.28±1.8	44.84±0.7	23.78±1.3	61.43±0.5	34.84±0.6	29.23 ± 0.7
Aux tasks [195]	✓	✓	✓	✓				73.4±2.0	52.08±1.4	23.40±1.5	63.62±1.6	42.27±1.2	24.79±2.2
Risk only	✓	✓				✓		74.90±1.7	50.25±1.1	22.56±1.2	66.22±1.2	45.26±0.8	24.47±1.7
Compass only	✓	✓					✓	75.08±1.5	50.55±1.0	22.49±1.1	67.32±1.7	45.74±1.0	23.54±1.7
Aux + risk	✓	✓	✓	✓	✓	✓		75.61±1.8	51.43±0.2	21.04±1.4	68.16±0.8	45.64±0.2	22.00±1.6
Aux+compass	✓	✓	✓	✓	✓		✓	75.63±1.2	52.60±1.6	23.17±1.2	67.94±1.4	45.76±1.0	23.78±2.0
Social tasks	✓	✓				✓	✓	76.6±1.8	52.81±1.2	20.47±0.4	68.35±0.5	45.83±0.5	21.72±1.2
Social + Aux tasks	✓	✓	✓	✓	✓	✓	✓	77.24±1.1	55.23±1.4	19.50±1.0	70.16±1.1	47.60±1.0	22.09±1.3

Table 5.1: Social Navigation evaluation on Gibson4+ and HM3D-S. For each model are listed the type of input data it uses (*Sensors* column) and, eventually, what kind of self-supervised *Aux tasks* or *Social tasks* the model employs. The metrics reported are *Success* rate, *SPL* and Human-Collisions Rate (*H-collisions*).

Datasets and training procedure. We performed our experiments using Gibson4+ and HM3D-S, a new dataset based on HM3D and adapted for social navigation ³. Gibson4+ contains 86 high-rated scenes taken from the original Gibson dataset [185]. For training, we used 64 scenes, while 8 and 14 environments were used for validation and test, respectively. HM3D-S is a dataset for Embodied Social Navigation that we generated on top of HM3D [133]. It consists of 900 scenes (800 used for training, 30 for validation, and 70 for test) with, on average, a larger walkable surface compared to Gibson4+. We have generated 8M episodes for the training set (10k per scene), 450 for the validation set (15 episodes per environment), and 490 for the test set (7 episodes per environment). Each episode is obtained by selecting a starting point and a goal point for the agent from the navigable area of the environment (such that it exists a navigable path from one to the other). Pedestrians are included as in [196]; namely, each person is positioned on a starting point and navigates back and forth to an endpoint with a random linear velocity between 0.45 and 0.5m/s. On Gibson4+, we trained each model for ≈ 100 M steps of experience (2.5 days training). On HM3D-S, we fine-tuned our models for ≈ 40 M steps (1-day training) starting from the final checkpoint obtained on Gibson4+. This was done to reduce the computational cost of each training.

5.4.1 Results

Evaluation Metrics. We used standard evaluation metrics for Point Goal Navigation such as *Success Rate* and *Success weighted by Path Length* (SPL) to evaluate the efficacy of the policy [6]. To evaluate its safety properties, we used *Human Collision Rate*, which is the percentage of episodes that end with failure by hitting a person. We run all our experiments with five runs to assess the mean and standard deviation for every metric, as done in [196].

Baseline models. We compared our approach to two baseline models: the model used by [196] (called *Baseline*) and a version of our model that only uses a set of 3 self-supervised auxiliary tasks: 2 CPC|A tasks (respectively using 2 and 4 steps) and GID (4 steps) (called *Aux tasks*), starting from the one provided by [195]. *Baseline* only uses the depth channel as input. Since we believe that RGB input is fundamental for people recognition and trajectory prediction, we also experimented with an adapted version of the model that uses all the information from RGB-D frames (called *Baseline+RGB*).

³Dataset, splits, code, and features will be publicly released.

Socially-aware models and auxiliary tasks. Firstly, to evaluate each social task contribution, we experimented with single-GRU models equipped with just one Socially-Aware task at a time (i.e., *Risk-only* or *Compass-only*). Since the tasks deal with two different aspects of social navigation (short-range and long-range), we then tried to combine them in a two-GRU model (referred as to *Social tasks*). Finally, we combined our approaches with the self-supervised auxiliary tasks presented in [195], which reported state-of-the-art performance on PointGoal Navigation. We have thoroughly investigated the benefit of combining them with single social tasks (*Aux+Risk* and *Aux+compass*), as well as combining them all in our final model (*Social+Aux tasks*). We now discuss our results and highlight the main takeaways.

Performance analysis and comparison to prior work. Table 5.1 reports the social navigation performance (on the test set) for both Gibson4+ and HM3D-S. In both cases, *Aux tasks* appears as the strongest of our baselines (highest SPL and lowest Human-Collision for both datasets), reaching comparable performances to single social task models while having a higher SPL. Our initial hypothesis that integrating the Baseline with an RGB signal could benefit performances was partially supported by the results on Gibson4+. However, the trend shifted on HM3D-S. This happens because of the higher quality of scene reconstruction in Gibson4+ (scenes have been manually rated and are among the best in the original Gibson dataset). Comparatively, HM3D-S has more reconstruction errors that, while leaving depth-only policies unaffected, may impair the performance of RGB-enabled models. Moreover, we notice that both models that use just one Socially-aware task perform similarly on Gibson4+ (sub 0.5% of difference between metrics). However, this changes on HM3D-S, where Compass-only slightly outperforms Risk-only (+1.1% Success, -0.93% h-collisions). This difference is expected since the social compass task explicitly aims to deal with long-range social information. Being HM3D scenes larger in size, the Social Compass role becomes more important.

Adding self-supervised tasks significantly increases SPL and Success performances (both for single-task and all Social-tasks models). It also appears to positively affect Human Collision when combined with Risk (-1.52% in Gibson4+, -2.47% in HM3D-S). We hypothesize that self-supervised tasks, since they are either action-based contrastive tasks (CPC|A) or tasks that try to retrieve the inverse dynamics of navigation (GID), help socially-aware models to have smoother trajectories thanks to a more accurate linear and angular velocity dialing. This claim will be substantiated by the fine-grained analysis reported in the next section. Overall, the best results are obtained

by combining all tasks together in the same model.

5.4.2 Fine-grained evaluation

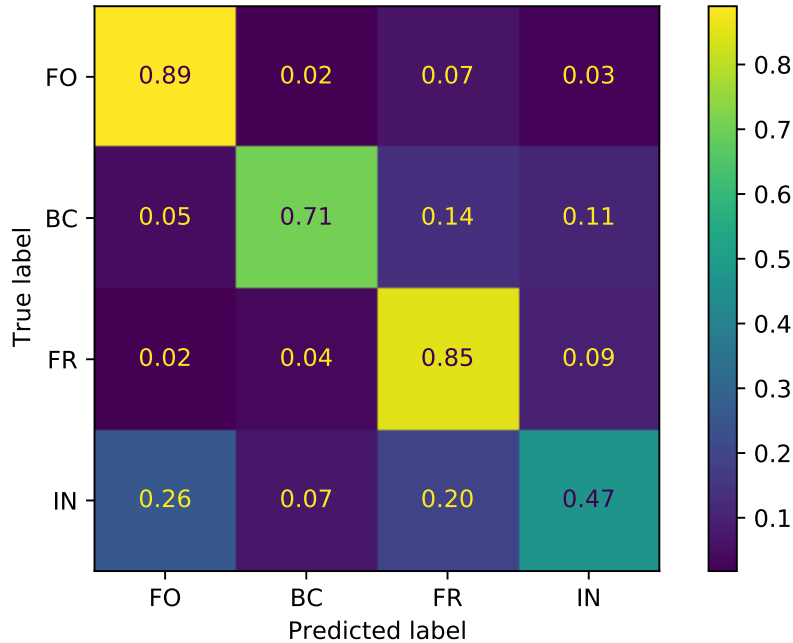


Figure 5.5: Confusion matrix obtained from the questionnaire, normalized for each true label.

Encounter evaluation (human studies) For our fine-grained evaluation we used a fixed set of hard-coded *Inclusion Rules*, based on the relationship between agent and people’s dynamics, to determine to which class an encounter belongs to. To investigate possible overlaps/ambiguity between classes, we conducted a survey to collect classification data from 15 human subjects through a multiple-choice questionnaire.

After explaining each encounter class through an informal definition, each participant was asked to classify 200 encounters. Each encounter was presented as a GIF with the target person highlighted in red (for some examples, see the video file).

Questionnaire results. Since we are mainly interested in per-class ambiguity, we show in Figure 5.5 the confusion matrix with row-wise normalization (per ground-truth label).

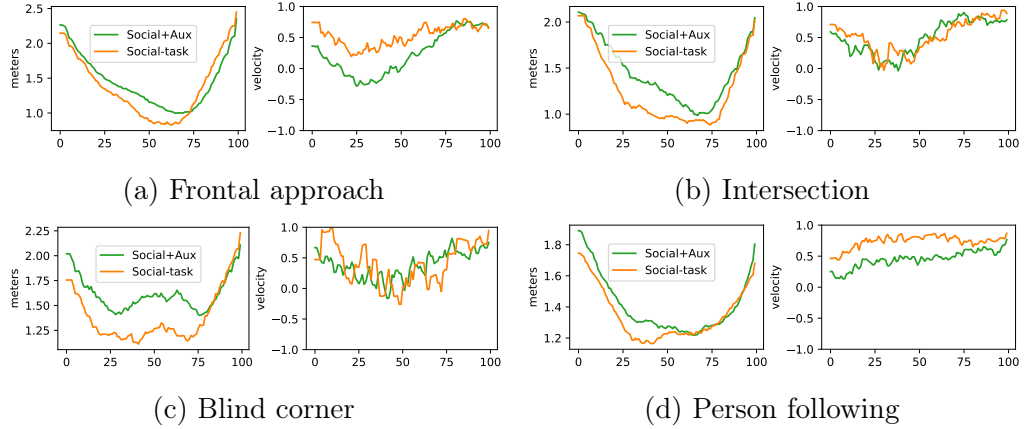
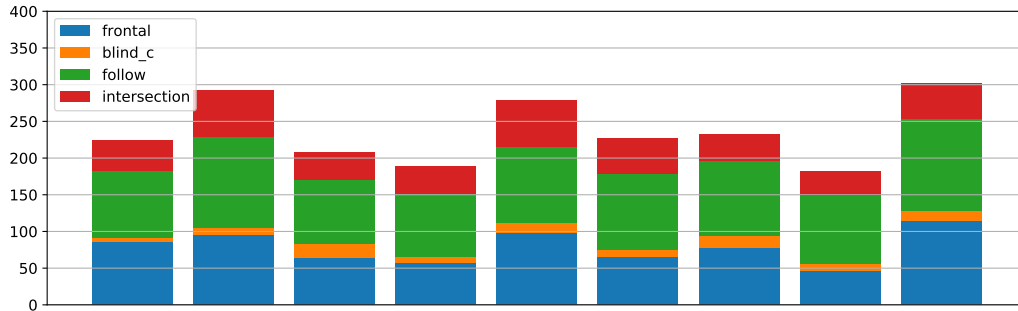


Figure 5.6: ALV and AD curves of top models for each encounter class.

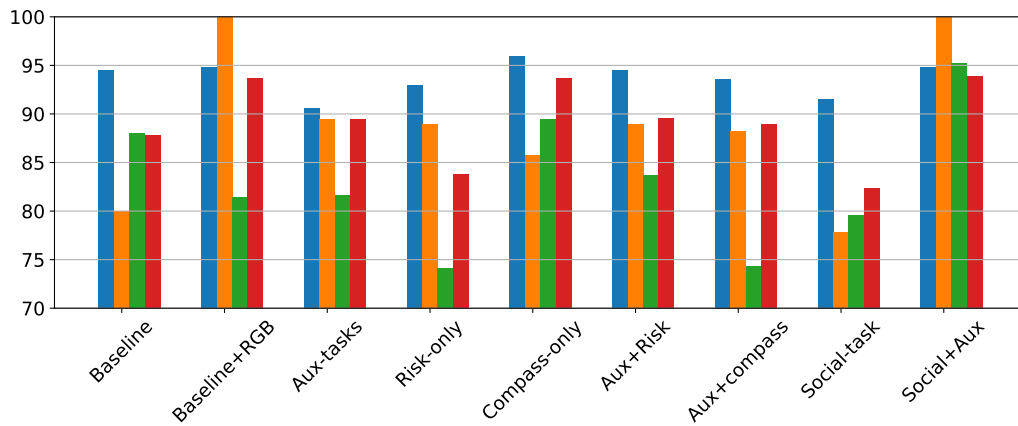
On average, the participants can clearly distinguish encounters belonging to Person following, Blind corner and Frontal approach. However, the intersection class is ambiguous and an encounter belonging to this class can be sometimes mistaken for either Frontal Approach or Person Following. This is expected since the main difference between these classes is the angle between the agent’s and person’s trajectories and the intersection class is in between the other two. This makes it hard for the naked eye to distinguish an intersection encounter unless the angle of intersection is very close to $\frac{\pi}{2}$. This problem could be mitigated by narrowing the Δ_{slack} angle but this causes far too many intersections to be excluded from any class. We decided to prioritize coverage over disambiguation by choosing Δ_{slack} to be just slightly lower than $\frac{\pi}{4}$.

We report the results obtained on the Gibson4+ dataset by applying our evaluation protocol (defined in Section 5.2) to understand better how each model operates and their attitude towards social interactions. Figure 5.7 summarizes the statistics, in terms of number of encounters and ESR, collected for each encounter class by different models during 500 randomly sampled validation episodes.

Policy behavior analysis. Looking at the relationships between the number of encounters and ESR figures, there seem to be two types of policies: a first group with a high number of encounters and a high ESR, and a second group that tends to avoid encounters and has low/medium ESR. Those two types describe different approaches to social navigation: either risking to interact to access potentially more efficient routes to the goal or keeping a safe distance from humans (for example, prioritizing less populated areas,



(a) Number of encounters



(b) Encounter Survival Rate (ESR)

Figure 5.7: Number of encounters vs ESR for all the models.

waiting for people to move away before crossing a room).

An example of a policy that avoids encounters is *Risk only*, which has one of the lowest ESR for Following (74.11%) and Intersection class (86.85%) and the lowest number of encounters (188 in total). The opposite is true for *Compass only*, which has a high overall ESR for every encounter class and one of the highest numbers of encounters (279 in total). It is interesting to note how the two best-performing policies, *Social tasks* and *Social+Aux*, adopt each of these different approaches while remaining comparable in coarse-grained metrics.

Reacting to sudden danger. A critical ability that a social policy must possess is the capability to react to immediate and sudden danger in situations with limited visibility. The class of encounters that better represents this is the *blind corner* class. To investigate how our two best models react compared to the baseline, in Figure 5.8, we plotted the AD and ALV values at

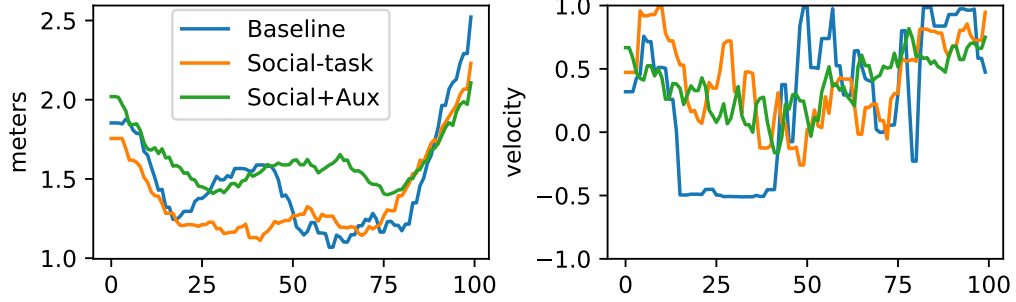


Figure 5.8: Average Distance (on the left) and Linear Velocity curves (on the right) for the Blind-Corner class.

a percentage of completion of all blind corner encounters. We can notice how the ALV curves are not smooth, reflecting uncertainty and high risk. However, while the baseline needs to brake and backtrack (≈ -0.5 ALV between 20% and 40% of the episodes), the other models tend to maintain a positive and proportionate ALV velocity throughout the episode. We can also notice that the *Social+Aux* ALV curve is smoother than the one of *Social tasks*. The same phenomenon is true for each single socially-aware task model and their self-supervised task counterpart (see supplementary material). This supports the claim that self-supervised tasks provide a smoothing effect on action dynamics under uncertainty.

In Figure 5.6, we show the AD and ALV curves for the *Social-tasks* and *Social+Aux* models.

General class trends. AD curves are shaped similarly across different classes. However, Person Following has a higher minimum ($\approx 1.4\text{m}$) since the agent and the person are heading in the same direction and not towards each other. This means that the agent does not need to slow down and then accelerate again after avoiding a collision but can maintain a cruising velocity that allows them to maintain a safe distance. This trend is shown by the corresponding ALV curves that are not bell-shaped but increase over time and are always above 0.

Top models comparison. In general, *Social+Aux* maintains a higher AD, for most of the time, than *Social-tasks* in all classes. This reflects the results obtained on ESR metrics since maintaining a safe distance deals higher chances of survival.

In turn, in situations where the agent and the person have parallel trajectories (i.e., during Person following and Frontal approach), this policy has a

lower ALV (in particular for the Frontal encounter, we have a negative ALV peak, suggesting that this policy generally yields and lets the person move out of the way). This happens because, in such situations, lower velocity and yielding is usually a safer strategy. Instead, during Blind corner and Intersection, a viable option could be accelerating and getting away from danger as fast as possible.

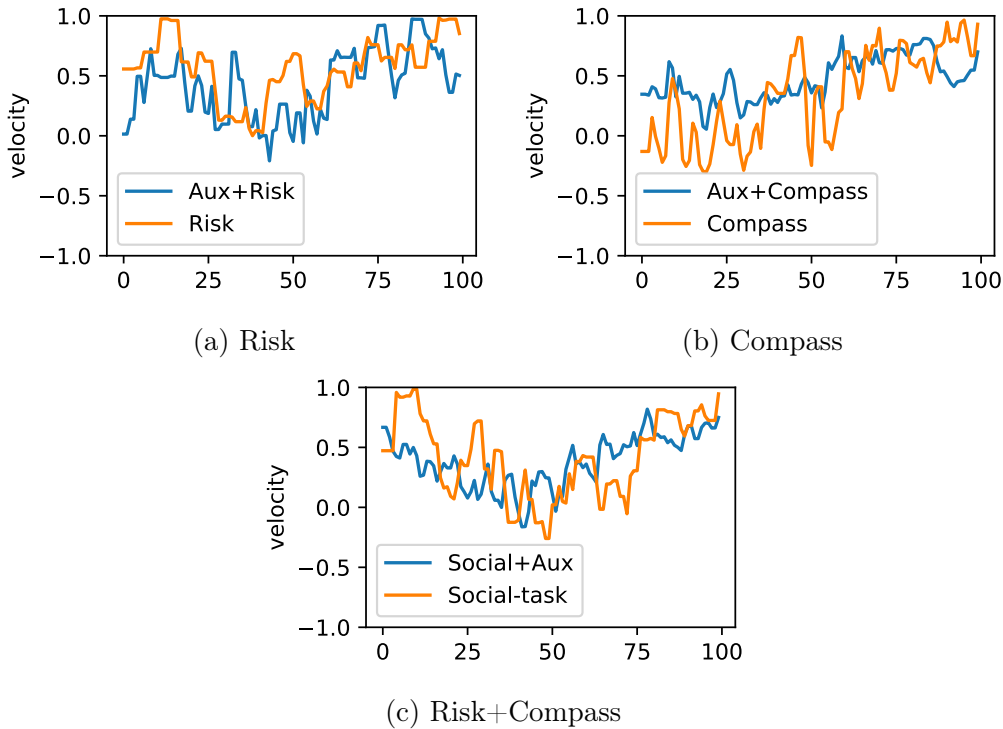


Figure 5.9: Comparison of Blind-corner ALV curves with each Socially-Aware task combination and the respective model with aux tasks added.

Smoothing effect of self-supervised auxiliary tasks. In Figure 5.9, we compared Blind-corner ALV curves for each policy that uses either single or a combination of Socially-Aware tasks and the corresponding models with self-supervised auxiliary tasks. We can notice the smoothing effect in every instance, even though it is less noticeable for Risk-only. It is particularly effective on Compass-only, a policy that behaves poorly on Blind-corner (reflected by the corresponding ESR value) because it considers just the long-range aspects of social information and not the short-range ones, particularly useful when dealing with unexpected danger.



Figure 5.10: Two success cases: a frontal encounter (top) and an intersection encounter (bottom).

Qualitative results. Figure 5.10 shows two qualitative examples of successfully managed encounters in Gibson4+. The first example (on top) depicts a Frontal encounter. After seeing people (first frame), the agent moves to the side and yields, letting them move away (second frame). Finally, the agent can reach the goal (third frame). In the second example (an intersection encounter), The agent sees a pedestrian (first frame), then it yields, letting the pedestrian pass (second frame), and finally, it continues on its path.

5.5 Conclusion

We introduced a model for Embodied Social Navigation based on two *Socially-Aware tasks*. Our experiments show that exploiting social signals, alone or in combination with self-supervised auxiliary tasks, is an effective strategy in complex and crowded scenarios. Our model can avoid the majority of encounters by using only Socially-Aware tasks. Furthermore, by combining Socially-Aware and auxiliary tasks [194], it can prevent human collisions in almost all the cases, despite a higher number of encounters. However, a major limitation of our setup, and more broadly of the Embodied Social Navigation task, resides in the simple movement of pedestrians. In future works we would like to focus on simulating more natural human behaviors and to experiment on sim2real domain transfer.

Chapter 6

Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions

In this chapter, we present an end-to-end framework [54] by which an agent can jointly learn perception and symbolic functions while supervising only the final output (**R4**). This is the first architecture able to learn both perception and symbolic functions in an end-to-end fashion. As stated in Chapter 1, this kind of architecture could potentially be very useful in Embodied AI to directly learn symbolic properties about environments during exploration.

6.1 Introduction

¹Neuro-Symbolic (NeSy) Systems combine deep neural networks and symbolic reasoning so that learning and reasoning can occur in a symbiotic fashion. The fundamental goal of NeSy systems is to incorporate and potentially learn the symbolic rules while still exploiting neural networks (NNs) for interpreting perception and guiding exploration in the combinatorial search space. In general, a NeSy system can be seen as a composition of *perception functions* and *symbolic functions*. Perception functions map perception, usually represented as real-valued tensors, to symbols, whereas symbolic functions map symbols to other symbols. The first challenge to any NeSy system is to reconcile the dichotomy between the intrinsically discrete nature of symbolic reasoning and the implicit continuity requirement of gradient descent-based learning methods. Recent works have tried to resolve this

¹A. Daniele, **T. Campari**, S. Malhotra, L. Serafini. "Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions". Under submission.

problem by exploiting different types of continuous relaxations to logical rules. However, with few exceptions, most such works assume the symbolic functions to be given a priori, and they use these functions to guide the training of a perception function, parameterized as a NN. A key challenge to such systems is the lack of a method capable of performing symbolic manipulations and meaningfully associating symbols to perception inputs, also known as the *Symbol Grounding Problem* [75].

In this chapter, we introduce the concept of *NeSy-function*, i.e., a composition of a set of perception and symbolic functions. Moreover, we propose *Deep Symbolic Learning (DSL)*, a framework that can jointly learn perception and symbolic functions while supervised only on the NeSy function. This is done by introducing policy functions, similar to Reinforcement Learning (RL) [161], within the neural architecture. The policy function chooses internal symbolic representations to be associated with the perception inputs based on the confidence values generated by the neural networks. The selected symbols are then combined to form a unique prediction for the NeSy function, while their confidences are interpreted under fuzzy logic semantics to estimate the confidence of such a prediction. Moreover, DSL can learn symbolic functions by applying the same policy to select their outputs. The key contributions of DSL are:

- *Learning the symbolic and the perception function through supervision only on the NeSy function.* To the best of our knowledge, DSL is the first NeSy system that can simultaneously learn symbolic and perception functions in an end-to-end fashion, from supervision only on their composition. It has been shown that previous such claims [173] contained some form of label leakage leading to supervision on the individual perception functions [34], and the system completely fails (with 0% accuracy on visual-sudoku task) when supervision on the perception function is removed. Furthermore, later works on this idea rely on clustering-based pre-processing [165] and do not constitute an end-to-end system.
- *Symbol Grounding Problem (SGP)* refers to the problem of associating symbols to abstract concepts without explicit supervision [75] on this association. The SGP is considered a major prerequisite for intelligent agents to perform real-world logical reasoning. Recent works [34] have also provided extensive empirical evidence on the non-triviality of this task, even on the simplest of problems. In DSL we can create internal (interpretable) symbolic representations that are then associated to perception inputs (e.g., handwritten digits) while getting supervision

only on higher order operations (e.g., the sum of the digits). Furthermore, unlike previous works [165], DSL does not rely on clustering based pre-processing. This is important as such pre-processing informs the system about the number of symbols, whereas DSL can infer the number of required symbols and create meaningful associations between symbols and perception inputs.

- *Differentiable Discrete Choices.* DSL is the first NeSy architecture that provides a method for making discrete symbolic choices within an end-to-end differentiable architecture. It achieves this by exploiting a *policy function* that given confidence values on an arbitrarily large set of symbols, is able to discretely choose one of them. Furthermore, the policy function can be changed to exploit varying strategies for the choice of symbols.

Finally, we provide extensive empirical verification of the aforementioned claims by testing DSL on three different tasks. Firstly, we test our system on the MNIST [100] Sum task [109], with an extension that no prior knowledge is given on the addition rules (see Example 1). We also test DSL with no prior information on the number of required internal symbols. DSL is able to select the required number of symbols and correctly associate them to perception inputs while learning the summation rules. DSL provides competitive results, even in comparison to systems that exploit prior knowledge. In the second experiment (see Example 2), we further generalize the first problem by adding an additional handwritten image of an elementary operation ($+$, $-$, \times , \div) to the two MNIST digits, with the label being the result of the operation on the digits. DSL is correctly able to learn the perception functions recognising the digits and the operations, and is able to learn the symbolic rules for each of the elementary operations. Finally, in the third experiment (see Example 3), we provide a sequence of handwritten digits of 0's and 1's, and the task is to learn the perception functions for recognizing the digits and to learn the symbolic rule for parity of the sequence. DSL is able to learn both the perception function for recognizing digits and the symbolic function for parity.

NeSy has emerged as an increasingly exciting field of AI, with several directions [22]. Approaches like Logic Tensor Networks [15] and Semantic-Based Regularization [60] encode logical knowledge into a differentiable function based on fuzzy logic semantics, which is then used as a regularization in the loss function. Semantic Loss [187], also aims at guiding the NN training through a logic-based differentiable regularization function based on probabilistic semantics, which is obtained by compiling logical knowledge into a Sentential Decision Diagram (SDD) [55]. In comparison to DSL these approaches assume that the symbolic function is already given and is not learned

from data. Furthermore, the symbolic function is only used to guide the learning of the perception function and does not influence the NN predictions at test time.

A parallel set of approaches incorporates NN's as atomic inputs to the conventional symbolic solvers. DeepProbLog [109], a neural extension to ProbLog [26], admits neural predicates that provide the output of an NN, interpreted as probabilities. The system then exploits SDDs enriched with gradient semirings to provide an end-to-end differentiable system for learning the NN and the program parameters simultaneously. Recent works have aimed at providing similar neural extensions to other symbolic solvers. DeepStochLog [182] and NeurASP [193] provide such extensions to Stochastic Definite Clause Grammars and Answer Set Programming respectively. In comparison to the regularization-based approaches, these approaches are able to exploit the symbolic function at the inference time. However, they also assume the symbolic function to be given. NeSy methods like NeuroLog [167], ABL [53] and ABLSim [81] are based on abduction-based learning framework, where the perception functions have supervision on assigning symbolic labels to perception data. However, the reasoning framework provides additional supervision to make the perception output consistent with the knowledge base i.e., the symbolic function.

Another paradigm of NeSy integration consists of works that aim at learning the symbolic function, with either no perception component or with supervision on the perception function. Neural Theorem Prover [140] uses soft unification to learn symbol embeddings to correctly satisfy logical queries. Logical Neural Networks [139] is a NeSy system that creates a 1-to-1 mapping between neurons and elements of a logical formulae. Hence, treating the entire architecture as a weighted real-valued logic formula. SATNet [173] aims to learn both the symbolic and perception functions. It does so by encoding MAXSAT in a semi-definite programming based continuous relaxation, and integrating it into a larger deep learning system. However, it has been shown that it can only learn the symbolic function when supervision on perception is given [34]. [165] extends SATNet to learn perception and symbolic functions, aiming at resolving the symbol grounding problem in SATNet. This extension relies on a pre-processing pipeline that uses InfoGAN [45] based latent space clustering. Besides not being end-to-end, their method assumes that the number of symbols (i.e., the number of digits in their experiments) is given a priori. [13] is another approach that exploits latent space clustering for extracting symbolic concepts. Furthermore, they assume the number of symbols and the logical rules to be given a priori. In DSL, only an upper bound needs to be provided on the number of required symbols. If the amount of symbols provided is higher than the correct one, it learns to ignore the

additional symbols, mapping the perceptions to only the required number of symbols.

Most of the NeSy systems in literature have distinct perception and symbolic components. To the best of our knowledge, none of these systems can learn both the components from supervision provided only on their composition. In DSL, we provide an approach that is able to learn both the symbolic functions and the perception functions separately, from supervision only on their composition. Furthermore, the symbols required to create the rules are created internally and are associated to perception within a unique NN learning pipeline. In comparison to the SOTA NeSy methods, DSL is the first end-to-end NeSy system to resolve a non-trivial instance of both the symbol grounding problem and rule learning from perception.

6.2 Background

Notation. We denote sets with math calligraphic font and its elements with corresponding indexed lower case letter, e.g., $\mathcal{S} = \{s_i | \forall i \in \mathbb{N}, 0 < i \leq k\}$, where $k = |\mathcal{S}|$ is the cardinality of the set. Tensors are denoted with capital bold letters (e.g. \mathbf{G}) and the $[\dots]$ operator is used to index values in a tensor. For instance, given a matrix $\mathbf{G} \in \mathbb{R}^{10 \times 10}$, the element $\mathbf{G}[1, 2]$ corresponds to the entry in row 1 and column 2 of \mathbf{G} . Similar to python syntax, we introduce the colon symbol for indexing slices of a tensor. As an example, $\mathbf{G}[1, :]$ corresponds to the vector $\langle \mathbf{G}[1, 1], \dots, \mathbf{G}[1, 10] \rangle$. We use a bar on top of functions and elements of a set to denote a tuple of functions and elements, respectively. For instance, $\bar{s} = \bar{f}(\bar{x})$ is equivalent to:

$$(s_1, \dots, s_n) = (f_1(x_1), \dots, f_n(x_n))$$

Note that the length n of the tuple is omitted from the bar notation since it will always be clear from the context.

Fuzzy Logic. Fuzzy Logic is a multi-valued generalization of classical logic, where truth values are reals in the range $[0, 1]$. In this work, we will only be dealing with conjunctions, which in fuzzy logic are interpreted using *t-norms*. A t-norm $t : [0, 1] \times [0, 1] \rightarrow [0, 1]$ is a function that, given the truth values t_1 and t_2 for two logical variables, computes the truth value of their conjunction. In this chapter, we will exploit Gödel t-norm, which defines the truth value of a conjunction as the minimum of t_1 and t_2 .

6.3 Problem Definition

Our approach to NeSy can be abstractly described as the problem of jointly learning a set of perception and symbolic functions providing supervision only on their composition. We define \mathcal{X} to be the space of possible perception inputs. Given a finite set \mathcal{S} of discrete symbols, a *perception functions* $f : \mathcal{X} \rightarrow \mathcal{S}$ maps from \mathcal{X} to symbolic output in \mathcal{S} . We define a *symbolic function*, $g : \mathcal{S}^n \rightarrow \mathcal{S}$, that maps an n -tuple of symbols to a single output symbol. We will also consider g with a typed domain, i.e., given some sets of symbols $\mathcal{S}_1, \dots, \mathcal{S}_n$ and \mathcal{S} , g could map from $\mathcal{S}_1 \times \dots \times \mathcal{S}_n$ to \mathcal{S} . Finally, we define a *NeSy functions* $\phi : \mathcal{X}^n \rightarrow \mathcal{S}$ as a composition of perception and symbolic functions. In this chapter, we will provide supervision only on the NeSy-function through a training set Tr of the form $Tr = \{(\bar{x}_i, y_i)\}_{i=1}^m$, where $y_i = \phi(\bar{x}_i)$ and m is the dimension of the training set. The goal is learning both the NeSy function and its components. NeSy-functions can constitute arbitrary compositions of symbolic and perception functions. In this chapter we consider two such cases, namely *Direct Nesy function*, and *Recurrent NeSy function*.

Definition 6.3.1 (Direct NeSy function). Let $g : \mathcal{S}_1 \times \dots \times \mathcal{S}_n \rightarrow \mathcal{S}$ be a symbolic function and $f_i : \mathcal{X} \rightarrow \mathcal{S}_i$, for $i = 1, \dots, n$ be n perception functions. A *Direct NeSy-function* is defined as the composition of g with the f_i

$$\phi(x_1, \dots, x_n) = g(f_1(x_1), \dots, f_n(x_n)) \quad (6.1)$$

Example 1 (Direct NeSy-function - Sum task). Let \mathcal{S}_1 and \mathcal{S} be the following set of symbols: \mathcal{S}_1 are the integers from 0 to 9 and \mathcal{S} is the set of integers from 0 to 18. Let us have a training set, consisting of tuples (x_1, x_2, y) where x_1 and x_2 are images of handwritten digits and y is the result of adding the digits x_1 and x_2 . Our goal is to learn the Direct NeSy-function:

$$\phi(x_1, x_2) = g(f(x_1), f(x_2))$$

where f is handwritten digit classifiers. Hence, our goal is to learn f and g with supervision provided only on $g(\bar{f}(\bar{x}))$.

Example 2 (Direct NeSy-function - Multiop sum task). Let \mathcal{S}_1 , \mathcal{S}_2 , and \mathcal{S} be the following set of symbols: \mathcal{S}_1 are the digits from 0 to 9, \mathcal{S}_2 is the set of symbols with four mathematical operations, $\{+, -, \times, \div\}$ and \mathcal{S} is the set of integers from 0 to 81. Let us have a training set, consisting of tuples (x_1, x_2, x_3, y) where x_1 and x_3 are handwritten digits, x_2 is a handwritten

operator and y is the result of applying the operator to x_1 and x_3 . Our goal is to learn the Direct NeSy-function:

$$\phi(x_1, x_2, x_3) = g(f_1(x_1), f_2(x_2), f_1(x_3))$$

where f_1 are handwritten digit classifiers and f_2 is a classifier for handwritten mathematical operations. Again, our goal is to learn f_1 , f_2 , and g from $g(\bar{f}(\bar{x}))$ with no direct supervision on g and \bar{f} .

As a second type of composition we will consider Recurrent NeSy functions, i.e., NeSy functions defined recursively. In general, it is possible to define complex types of recurrent compositions involving multiple perception and symbolic functions. In this work we focus on the simplest version of Recurrent NeSy-function that takes in input a non empty sequence of perceptions and returns a single output symbol.

Definition 6.3.2 (Simple Recurrent NeSy-function). Let $g : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$ be a symbolic function and $f : \mathcal{X} \rightarrow \mathcal{S}$ be a perception function. Moreover, it is given an ordered list of perceptions $X = \{x^{(k)}\}_{k=1}^K$, with $x^{(k)} \in \mathcal{X}$. We define $X^{(k)}$ as the sequence of first k elements of X . A *Simple Recurrent NeSy-function* ϕ is defined recursively as:

$$\begin{aligned}\phi(X^{(k)}) &= g(f(x^{(k)}), \phi(X^{(k-1)})) \\ \phi(X^{(0)}) &= s^{(0)} \in \mathcal{S}\end{aligned}$$

Example 3 (Simple Recurrent NeSy-function). Let $\mathcal{S} = \{s_0, s_1\}$ be a set composed of two symbols, representing binary values, and $\phi(X)$ the Simple Recurrent NeSy function which represents the parity function, i.e., the function that returns s_0 if the number of s_1 in the sequence is even, s_1 if it is odd. $\phi(X)$ can be expressed in terms of a perception function f and a symbolic function g using previous definition of Simple Recurrent NeSy function: the f converts the perceptions in binary values, while the g represents the XOR operator, with $s^{(0)} = s_0$.

6.3.1 Policy Functions

In this chapter we will exploit the concept of *policy functions* inspired by Reinforcement Learning (RL). In RL, an agent has at its disposal a set of available actions, and at each time frame only one action can be performed. The goal is to select actions that maximize the expected reward. A strategy

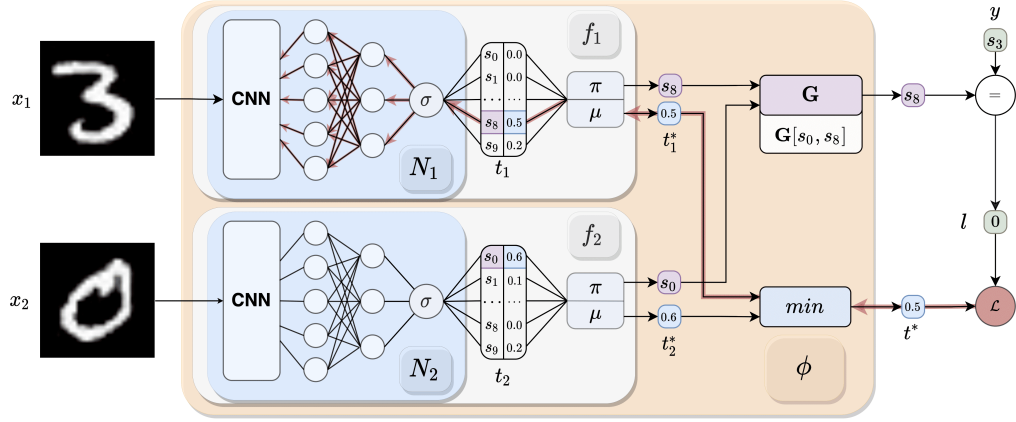


Figure 6.1: Architecture of Deep Symbolic Learning for the Sum task. Red arrows represent the backward signal during learning.

for choosing the actions, based on the current state of the system, is called a *policy*. In this work we consider two specific policies, namely the *greedy* and the ϵ -*greedy*, and we adapt them to the context of NeSy. In our setting, a *policy* selects a symbol instead of an action, and it is defined as a function $\pi : [0, 1]^{|S|} \rightarrow \mathcal{S}$ that, given a vector $t \in [0, 1]^{|S|}$, returns a symbol $s_i \in \mathcal{S}$. Intuitively, t is a vector of confidences returned by a neural network, which in our framework are interpreted as a vector of fuzzy truth values. Formally, t_i corresponds to the truth value of the proposition ($s_i = s^*$), where s^* is the correct (unknown) symbol. Moreover, we define the function $\mu : [0, 1]^{|S|} \rightarrow [0, 1]$ as the function that returns the truth value of the symbol chosen by the policy.

The *greedy policy* selects the symbol with highest truth value: $\pi(t) = \operatorname{argmax}_i t_i$. The function μ returns the corresponding truth value: $\mu(t) = \max_i t_i$. DSL exploit the differentiability of μ to indirectly influence the policy π , which is not differentiable. In the case of the greedy policy, by decreasing the highest confidence ($\mu(t)$), we reduce the chances for the current symbol to be selected again.

ϵ -*greedy* behaves like the greedy policy with probability $1 - \epsilon$, while it chooses a random symbol with probability ϵ . The advantage of ϵ -greedy over greedy is a better ability to explore the solutions space. In our experiments, we use ϵ -greedy during training, and greedy policy at test time.

6.3.2 DSL for Direct NeSy-functions

For sake of presentation, we first assume symbolic functions to be given, and our goal is to learn the perception functions. We will then extend DSL to learn also the symbolic function.

We first define the representation of the perception functions $f_i : \mathcal{X} \rightarrow \mathcal{S}_i$ and the symbolic function g . W.l.o.g., we assume that symbols in any set \mathcal{S}_i are represented by integers from 1 to $|\mathcal{S}_i|$. The symbolic function $g : \mathcal{S}_1 \times \dots \times \mathcal{S}_n \rightarrow \mathcal{S}$ is stored as a $|\mathcal{S}_1| \times \dots \times |\mathcal{S}_n|$ tensor \mathbf{G} , where $\mathbf{G}[s_1, \dots, s_n]$ contains the integer representing the symbolic output of $g(\bar{s})$. Every perception function $f_i : \mathcal{X} \rightarrow \mathcal{S}_i$ is modelled as $\pi(N_i)$, where $N_i : \mathcal{X} \rightarrow [0, 1]^{|\mathcal{S}_i|}$ is a neural network (NN), and $\pi : [0, 1]^{|\mathcal{S}_i|} \rightarrow \mathcal{S}_i$ is a *policy function*. For every $x \in \mathcal{X}$, $N_i(x)$ is an $|\mathcal{S}_i|$ -dimensional vector $\bar{t}_i \in [0, 1]^{|\mathcal{S}_i|}$ whose entries sum to 1. Intuitively, the l^{th} entry in \bar{t}_i represents the predicted truth value associated with the l^{th} symbol being the output of $f_i(x)$. The policy function π makes a choice and picks a single symbol from \mathcal{S}_i based on \bar{t}_i . In summary, our model is defined as:

$$\phi'(\bar{x}) = \mathbf{G}[\pi(N_1(x_1)), \dots, \pi(N_n(x_n))]$$

where ϕ' is the learned approximation of target function ϕ .

Example 4 (Example 1 continued). We assume the same setup as Example 1, with an addition that $f_i(x_i)$ is $\pi(N_i(x_i))$ (with $i \in 1, 2$), as presented above. Let the prediction of f_1 and f_2 be the integers 3 and 5 respectively. In this context, \mathbf{G} is a matrix that contains the sum of every possible pair of digits, so that $\mathbf{G}[i, j] = i + j$. Therefore, the prediction is: $\phi'(\bar{x}) = \mathbf{G}[3, 5] = 8$.

6.3.3 Learning the Perception Functions

In example 4, if one of the two internal predictions were wrong, then the final prediction 8 would be wrong as well. Hence, we define the confidence of the final prediction to be the same as the confidence of having both internal symbols correct simultaneously. In other words, we could consider the output $\phi'(\bar{x})$ of the model to be correct if the following formula holds for all perception input x_i :

$$(\pi(N_1(x_1)) = s_1^*) \wedge \dots \wedge (\pi(N_n(x_n)) = s_n^*) \quad (6.2)$$

where s_i^* is the (unknown) ground truth symbol associated to perception x_i . We interpret formula in Equation 6.2 using Gödel semantics, where the conjunctions are interpreted by the *min* function. We use t_i^* to denote the truth value (or the confidence) given by the NN for the symbol selected by π , i.e., $t_i^* = \mu(N_i(x_i))$. Hence, the truth value t^* associated to the final prediction

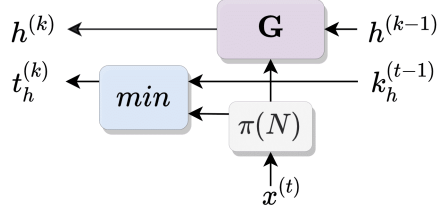


Figure 6.2: Architecture of Deep Symbolic Learning for the simple recurrent NeSy functions.

$\phi'(\bar{x})$ is given as:

$$t^* = \min_i t_i^* = \min_i \mu(N_i(x_i))$$

To train the model we use the binary cross entropy loss on the confidence t^* of the predicted symbol. If it is the right prediction, the confidence should be increased. In such a case, the ground truth label is set to one. If $\phi'(\bar{x})$ is the wrong prediction, the confidence should be reduced, and the label is set to zero. In summary, the entire architecture is trained with the following loss function:

$$\mathcal{L} = - \sum_{(\bar{x}, y) \in Tr} l \cdot \log(t^*) + (1 - l) \cdot \log(1 - t^*)$$

where $l = \mathbb{1}(\phi'(\bar{x}) = y)$, and $\mathbb{1}$ is the indicator function. The architecture is summarized in Figure 6.1, where we show an instance of DSL in the context of Example 1.

6.3.4 DSL for Recurrent NeSy-functions

Simple recurrent NeSy function is recursively defined as:

$$\begin{aligned} \phi'(X^{(k)}) &= \mathbf{G}[\pi(N(x^{(k)})), \phi'(X^{(k-1)})] \\ \phi'(X^{(0)}) &= \pi(\sigma(\mathbf{W}_0)) \end{aligned}$$

where $\mathbf{W}_0 \in \mathbb{R}^{|\mathcal{S}|}$ is the set of weights associated to the initial output symbol. Again, we define $t^* = \min_i t_i^*$ as the minimum among the truth values of the internally selected symbols. It is worth noticing the similarity between the DSL model and the equation 6.3.2. In general, a DSL model can be instantiated by following the same compositional structure of the NeSy function we want to learn, applying the policy when a value is expected to be symbolic. The architecture is presented in Figure 6.2.

6.3.5 Learning Symbolic Functions

So far we have assumed the symbolic function g to be given. We now lift this assumption and define a strategy for learning the g . The idea comes from a simple observation: for each tuple \bar{s} there exists exactly one output symbol $g(\bar{s})$. Note that the mechanism introduced to select a unique symbol from the NN output can be also used for selecting *propositional symbols*, i.e. static symbols that do not depend on the current perceptions. We use the policy functions on learnable weights, allowing to learn the symbolic rules directly from the data.

Formally, we define a tensor $\mathbf{W} \in \mathbb{R}^{|\mathcal{S}_1| \times \dots \times |\mathcal{S}_n| \times |\mathcal{S}|}$ as the weight tensor of \mathbf{G} . Note that the tensor shape is the same as \mathbf{G} , except for the additional final dimension, which is used to store the weights for all of the output symbols. The entry in \mathbf{G} corresponding to tuple \bar{s} is defined as:

$$\mathbf{G}[\bar{s}] = \pi(\sigma(\mathbf{W}[\bar{s}, :]))$$

where the softmax function σ and the policy π are applied along the last dimension of \mathbf{W} . The method is summarized by Figure 6.3.

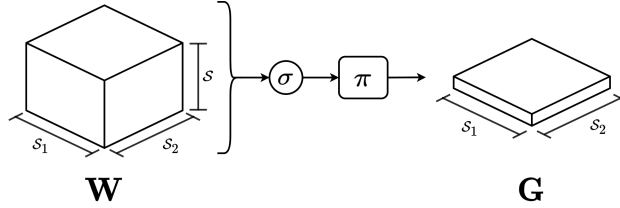


Figure 6.3: Tensor \mathbf{W} is used by the policy to generate tensor \mathbf{G} . This is done by applying the policy on the output dimension (vertical axes in the image), selecting a single output element for each pair of symbols $(s_1, s_2) \in \mathcal{S}_1 \times \mathcal{S}_2$.

Since the tensor \mathbf{G} is now learned, we need to consider the confidence associated with the choice of symbols in \mathbf{G} . The confidence of the final prediction is now defined as

$$t^* = \min(t_G^*, \min_i t_i^*)$$

where t_G^* is the confidence of the output symbol for the current prediction:

$$t_G^* = \mu(\sigma(\mathbf{W}[\bar{s}, :]))$$

with $\bar{s} = \pi(\bar{N}(\bar{x}))$ corresponding to the tuple of predictions made by the perception functions.

6.3.6 Gradient Analysis for the Greedy Policy

We analyze the partial derivatives of the loss function with respect to the truth values t_i^* . We consider only a single training sample, and assume that the policy is the greedy one.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial t_i^*} &= -l \frac{\partial \log(t^*)}{\partial t_i^*} - (1-l) \frac{\partial \log(1-t^*)}{\partial t_i^*} \\ &= -\frac{l}{t^*} \frac{\partial t^*}{\partial t_i^*} + \frac{1-l}{1-t_i^*} \frac{\partial t^*}{\partial t_i^*}\end{aligned}$$

Now, since t^* is the minimum of all $\{t_j^*\}_j$, the term $\frac{\partial t^*}{\partial t_i^*}$ is 1 if t_i is the minimum value in $\{t_j^*\}_j$ and 0 otherwise, reducing the total gradient to the following equation :

$$\frac{\partial \mathcal{L}}{\partial t_i^*} = \begin{cases} -\frac{l}{t^*} + \frac{1-l}{1-t^*} & i = \operatorname{argmin}_j t_j^* \\ 0 & \text{otherwise} \end{cases}$$

For each sample, only one confidence value t_i^* has a gradient different from zero, meaning that a single perception function is updated. This behavior is shown in Figure 6.1 by using red arrows to represent the backward signal generated by the backpropagation algorithm. The signal moves from the loss to f_1 , which corresponds to the symbol with lower confidence, and when it reaches the softmax function (σ), it is spread to the entire network. In DSL, we have not only interpretable predictions, but gradients are interpretable as well. Indeed, for each sample, there is a unique perception function f_i taking all the blame (or glory) for a bad (or good) prediction of the entire model ϕ' .

We evaluate our approach on different tasks where a combination of perception and reasoning is essential. Our goal is to demonstrate that: *i*) DSL can learn the NeSy function, while simultaneously learning the two components f and g , in an end-to-end fashion (MNIST sum); *ii*) The perception functions f_i learned on a given task are easily transferable to new problems, where the symbolic function g has to be learned from scratch, with only a few examples (MNIST Minus - One-Shot Transfer); *iii*) DSL can learn multiple symbolic functions at once (MultiOperation MNIST); *iv*) DSL can also be generalised to problems with a recurrent nature (MNIST visual parity).

Evaluation Metrics Our goal is to evaluate DSL's efficacy for learning the NeSy function ϕ , the perception functions f_i and the symbolic function g . However, this poses a problem: the symbols associated to perception inputs in DSL are internally generated and form a permutation invariant representation. Any permutation of the symbols leads to the same behavior of the model,

given that the same permutation is applied to the indices of the tensor \mathbf{G} . Hence, to evaluate the model on learning of g and f_i , we need to select a permutation that best explains the model w.r.t the “human” interpretation of symbols for digits. The problem is highlighted in Figure 6.4(left), where the confusion matrix of the MNIST digit classifier is introduced. Note that for each row (digit), only one column (predicted symbol) has a high value. The same is true for the columns. The network can recognize and distinguish the various digits, but the internal symbols are randomly assigned. This problem leads to difficulty in accessing the quality of DSL results, as we cannot easily interpret the mapping learned by the model. To obviate this problem, we calculate the permutation of columns of the confusion matrix which produces the highest diagonal values (Figure 6.4(right)). We then apply the same permutation on the confusion matrix and \mathbf{G} , allowing us to apply the chosen metrics for both digits (we use F1) and learned \mathbf{G} (accuracy on the learned rules). In all of our experiments, tensor \mathbf{G} is perfectly learned, i.e., we have an accuracy of 100% on the learned rules. For this reason, we will omit this metric in the next sections.

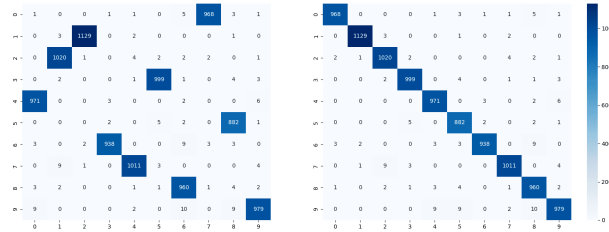




Figure 6.4: Confusion matrix for the MNIST digits: (left) before the permutation; (right) after permutation.

Implementation Details All the experiments were conducted with a machine equipped with an NVIDIA GTX 1070, with 8GB RAM. We used the Cuda toolkit 11.3, Python3.8, and PyTorch1.12.0. For digit classification, we use the same CNN as [109], which consists of two convolutional layers with MaxPool, followed by two linear layers. Between one layer and another, we used the ReLU activation function and, on the final features of the model, the SoftMax function to extract the truth values of the symbols. We used MadGrad [57] for optimization, with a learning rate of 0.1. Results are average over 5 runs. Finally, in all experiments, we used the ϵ -greedy policy with $\epsilon = 0.2$ during training and the greedy policy during inference.

	Accuracy (%)	F1 (%)	TE/#E
NAP	97.3 ± 0.3	-	-
DPL	97.2 ± 0.5	-	-
DStL	97.9 ± 0.1	-	-
DSL	98.8 ± 0.3	97.9 ± 0.6	1.35s/30
DSL-NB	97.9 ± 0.3	97.4 ± 0.5	1.39s/150
DSL-RN	98.8 ± 0.1	97.8 ± 0.2	21.65s/150

Table 6.1: Results obtained on the MNIST sum task. TE is the time required for 1 epoch, and #E the number of epochs of training. The SOTA methods are NeurASP (NAP), DeepProbLog (DPL), and DeepStochLog (DStL). SOTA results taken from [182].

MNIST Sum The first experiments are on the MNIST sum task as presented in Example 1. We are given a dataset consisting of triples (X, Y, Z) , where X and Y are two images of hand-written digits, while Z is the result of the sum of the two digits, e.g., (, , 8). The goal is to learn an image classifier for the digits and the function g which maps digits to their sum. We implemented three different variants of our approach: **DSL** is the naive version of DSL, where the two digits are mapped to symbols by the same perception function, and the correct number of digits is given a priori; **DSL-NB** is a version of **DSL** where we removed the two aforementioned biases: we use two different neural networks, N_1 and N_2 , to map perceptions to symbols, and the model is unaware of the right amount of latent symbols, with the neural network returning confidence on 20 symbols instead of 10; finally, in **DSL-RN** we use the same setup of **DSL**, but with a ResNet9 backbone instead of the CNN.

In table 1, we show that all variants of DSL have competitive performance w.r.t the state of the art [193, 182, 109]. Notice that all the SOTA methods receive a complete knowledge of the symbolic function g , while DSL needs to learn it, making the task much harder. Another important result is the accuracy of the DSL-NB method, which proves that DSL can work even with two perception networks and, most importantly, without knowing the right amount of internal symbols. Finally, results on DSL-ResNet show that DSL can work even with more complex perception networks.

MNIST Minus - One-Shot Transfer One of the main advantages of NeSy frameworks is that the perception functions learned in the presence of a given knowledge (g in our framework) can be applied to different tasks without retraining, just by changing the knowledge. For instance, after

learning to recognize digits from supervision on the addition task, methods like DeepProbLog can be used to predict the difference between two numbers. However, it is required for a human to create different knowledge bases for the two tasks. In our framework, the g function is learnable, and the mapping from perception to symbols does not follow human intuition (see Evaluation Metrics section). Instead of writing a new knowledge for the Minus task, we replace the tensor \mathbf{G} with a new one and learn it from scratch. In our experiment, we started from the perception function learned from the Sum task and used a single sample for each pair of digits to learn the new \mathbf{G} . We obtained an accuracy of 98.1 ± 0.5 after 300 epochs, each requiring 0.004s. Note that we did not need to freeze the weights of the f . Since the perception functions already produce outputs with high confidence, DSL applies changes mainly on the tensor \mathbf{G} .

MNIST MultiOperation We now generalize the MNIST Sum task by adding additional operators as perception inputs, as described in Example 2. The perceptions are two images from the MNIST dataset and a third symbol representing an operation in $\{+, -, \times, \div\}$. The operation’s images are generated from the EMNIST dataset [49], from which we extracted the images of letters A , B , C , and D , representing $+$, $-$, \times , and \div respectively. In order to have the same codomain for all operators, we take these additional steps: we necessitate in the dataset that for any instance with subtraction operator, i.e., “ $-$ ”, the second term of the operation is smaller than the the first; “ \div ” is interpreted as integer division (e.g. $(5 \div 2 = 2)$) and it requires the second term to be greater than 0; “ \times ” is interpreted as the product operation modulo 19. Notice that the hypothesis space consists of 19^{400} possible symbolic functions g . Furthermore, the correct g needs to be identified while simultaneously learning the NNs for digit and operator recognition.

We trained DSL on this task and obtained an accuracy of 96.9 ± 0.3 , with an F1 of 97.4 ± 0.4 on the digits and 98.2 ± 0.2 on the letters. We trained our model for 1000 epochs, with an epoch time of 1.29s. These results show that DSL can learn complex symbolic functions while simultaneously learning to map multiple perceptions over different domains.

MNIST Visual Parity We instantiate the model introduced in Figure 6.2 for the parity task (see Example 3). The perceptions are images of zeros and ones from the MNIST dataset, and the neural network is the same CNN used for the MNIST sum task (with 2 output symbols instead of 10).

Learning the parity function from sequences of bits is a hard problem for neural networks, which struggle to generalize to long sequences. Indeed,

it has been shown that conventional deep learning methods can not easily learn the parity function [152]. Note that, in our setting, the parity function corresponds to the symbolic function g , and learning the perception function is an additional sub-task added by us. We use sequences with 4 images during training and 20 on the test. In 1000 epochs (0.26s per epoch), DSL reaches an accuracy of 98.7 ± 0.4 , showing great generalization capabilities. Indeed, as in the other tasks, DSL learned perfectly the function g , which corresponds to the XOR function. Since the symbolic function always returns the right prediction, the errors made by the model depend only on the perception functions. If the perceptions are correctly recognized, the model works no matter the length of the sequence.

6.4 Limitations

The key limitation of DSL is scalability, as the dimension of the hypothesis space for the symbolic function g can become both complex and computationally intractable. For instance, in the MultiOperation setting, we tried to run the same experiments, using the multiplication instead of its module. In this case, DSL remained stuck in a local minimum, unable to learn the perception and symbolic functions: the perception network could not distinguish any digit, mapping all of them to the same symbol. Similar behavior was observed in the visual parity task, where DSL failed to escape from a local minima when the sequence length was larger than four.

6.5 Conclusion

We presented Deep Symbolic Learning, a NeSy framework for learning the composition of perception and symbolic functions. To the best of our knowledge, DSL is the first NeSy system that can create and map symbolic representations to perception while learning the symbolic rules simultaneously. A key contribution of DSL is the integration of discrete symbolic choices within an end-to-end differentiable neural architecture. For this, DSL exploits the notion of policy deriving from RL. DSL can learn the perception and symbolic functions while performing comparably to SOTA NeSy systems, where complete supervision on the symbolic component is given. In the future, we aim to extend DSL to problems with a larger combinatorial search space. To this end, we aim to consider factorized matrix representations for the symbolic function g , and its weight matrix W . Furthermore, we aim to generalize DSL to more complex perception inputs involving text, audio, and vision.

Chapter 7

Conclusion

This thesis makes several contributions to Embodied AI. We started with four different research questions (**R1**: *How can an agent exploit common-sense knowledge about the environments?* **R2**: *How can an agent reuse previously acquired knowledge about a specific environment?* **R3**: *How can an agent comply with social rules?* **R4**: *How can an agent acquire knowledge and common-sense rules?*), and we gave a possible solution to the questions through different works.

- In Chapter 2, we forced the agent to learn (and use) relations taken from common-sense knowledge (**R1**), in particular, the subtle relation between objects and rooms. To this end, we developed a model where a shared embedding is injected into a Scene-Memory Transformer. We evaluated our approach on the Matterport3D dataset and on a reduced version designed to cope with the high computing demand required by embodied agents. Our evaluation confirmed that the shared embedding systematically improves performance over baseline methods.
- In Chapter 3 and 4, we provided two possible solutions on how to reuse previously acquired knowledge about a specific environment (**R2**). In particular, in Chapter 3, we defined a modular architecture for the Object Goal Navigation task that takes advantage of the acquired knowledge, stored in an abstract model, to relocate the agent across different episodes and to localize the nearest occurrence of the goal object. We evaluated our method on Matterport3D with the Habitat simulator, and we found that this is effective. In Chapter 4, instead, we tackled a long-horizon task, MultiON, always with a modular architecture. This time, we recorded the acquired information in a semantic map. We evaluated our approach on the HM3D dataset, and we found that for

long-horizon tasks, storing knowledge in a map is sufficient to achieve good performance.

- In Chapter 5, we presented an agent that exploits common-sense knowledge (**R1**), by means of two Socially-Aware auxiliary tasks, and can navigate cluttered environments while being aware of the notion of risk (**R3**). We evaluated our agent on Matterport3D and HM3D. We found that the method is effective under the standard evaluation metrics for Embodied Social Navigation metrics. Furthermore, we conducted a fine-grained analysis to better understand the agent's performance when an encounter with a human happens. In this experiment, our model is more aware of human presence and can avoid dangerous encounters.
- In Chapter 6, we presented an end-to-end framework that can simultaneously learn symbols from perceptions and symbolic functions. This is a preliminary work, but in the future a similar approach can be applied also in an Embodied agent, to directly learn how to map perceptions to symbols and to learn common-sense knowledge about an environment.

Future work may focus on reducing the gap between simulation and real-world performance for trained agents. It can be challenging to transfer policies learned in simulation to a real robot, so one potential direction is to conduct experiments in both simulation and the real world to improve transferability. Another possibility is to develop a general agent that can receive and execute instructions in natural language. This agent would be able to handle a wide range of tasks, such as the example given: "Go to the kitchen, grab a Coke from the fridge, and bring it to me." To accomplish this, the instruction could be transformed into an interpretable plan that the agent can understand, such as:

- Go to <room> (in this case, the kitchen)
- Find <object> (the Coke)
- Grab <object>
- Go to <coordinate> (initial coordinate)

This plan involves several different capabilities that have been addressed in previous work, including Embodied Social Navigation ("Go to <coordinate>") and Object Goal Navigation ("Find <object>"). Tools like ChatGPT could convert natural language instructions into a plan like the example above. By developing a general agent with this type of functionality, it may be possible to solve a vast array of tasks.

Bibliography

- [1] 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? In *RA-L*, pages 6670–6677, 2020. 5
- [2] Kush Agrawal. To study the phenomenon of the moravec’s paradox. *arXiv preprint arXiv:1012.3148*, 2010. 3
- [3] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 5
- [4] Diego Aineto, Sergio Jiménez Celorrio, and Eva Onaindia. Learning action models with minimal observability. *Artificial Intelligence*, 275:104–137, 2019. 33
- [5] Ziad Al-Halah, Santhosh Kumar Ramakrishnan, and Kristen Grauman. Zero experience required: Plug & play modular transfer learning for semantic visual navigation. In *CVPR*, pages 17031–17041, 2022. 48, 51
- [6] 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. 14, 15, 17, 32, 38, 58, 74, 76, 85
- [7] 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 *CVPR*, pages 3674–3683, 2018. 15, 32, 49, 76
- [8] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, pages 39–48, 2016. 50
- [9] Georges S. Auude, Brandon D. Luders, Joshua M. Joseph, Nicholas Roy, and Jonathan P. How. Probabilistically safe motion planning to avoid dynamic

- obstacles with uncertain motion patterns. *Autonomous Robots*, 35(1):51–76, 2013. 76
- [10] Ronald C Arkin, Ronald C Arkin, et al. *Behavior-based robotics*. MIT press, 1998. 3
- [11] Masataro Asai. Unsupervised grounding of plannable first-order logic representation from images. In *ICAPS*, 2019. 33
- [12] Masataro Asai and Alex Fukunaga. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*, 2018. 33
- [13] Yaniv Aspis, Krysia Broda, Jorge Lobo, and Alessandra Russo. Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*, pages 421–431, 8 2022. 98
- [14] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. In *ICLR*, 2017. 47
- [15] Samy Badreddine, Artur d’Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artif. Intell.*, 303:103649, 2022. 97
- [16] Aayush Bansal, Yaser Sheikh, and Deva Ramanan. Pixelnn: Example-based image synthesis. *arXiv preprint arXiv:1708.05349*, 2017. 54
- [17] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *CORL*, pages 420–429. PMLR, 2020. 50
- [18] Dhruv Batra, Angel X Chang, Sonia Chernova, Andrew J Davison, Jia Deng, Vladlen Koltun, Sergey Levine, Jitendra Malik, Igor Mordatch, Roozbeh Mottaghi, et al. Rearrangement: A challenge for embodied ai. *arXiv preprint arXiv:2011.01975*, 2020. 46, 48
- [19] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. Objectnav revisited: On evaluation of embodied agents navigating to objects. *arXiv preprint arXiv:2006.13171*, 2020. vi, 31, 32, 49, 52, 53, 76
- [20] Michael Beetz, Lorenz Mösenlechner, and Moritz Tenorth. Cram—a cognitive robot abstract machine for everyday manipulation in human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1012–1017. IEEE, 2010. 33

- [21] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*, pages 3–19. Springer, 2011. 76
- [22] Tarek R. Besold, Artur d’Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2021. 97
- [23] Margaret A Boden. 4 gofai. *The Cambridge handbook of artificial intelligence*, page 89, 2014. 3
- [24] Blai Bonet and Hector Geffner. Learning first-order symbolic planning representations from plain graphs. *CoRR*, abs/1909.05546, 2019. 33
- [25] Rodney A Brooks. Elephants don’t play chess. *Robotics and autonomous systems*, 6(1-2):3–15, 1990. 3
- [26] Maurice Bruynooghe, Theofrastos Mantadelis, Angelika Kimmig, Bernd Gutmann, Joost Vennekens, Gerda Janssens, and Luc De Raedt. Problog technology for inference in a probabilistic first order logic. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 719–724, 2010. 98
- [27] Tommaso Campari, Paolo Eccher, Luciano Serafini, and Lamberto Ballan. Exploiting scene-specific features for object goal navigation. In *European Conference on Computer Vision*, pages 406–421. Springer, 2020. 13, 32, 74, 76
- [28] Tommaso Campari, Leonardo Lamanna, Paolo Traverso, Luciano Serafini, and Lamberto Ballan. Online learning of reusable abstract models for object goal navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14870–14879, 2022. 29, 76
- [29] Enrico Cancelli, Tommaso Campari, Luciano Serafini, Angel X Chang, and Lamberto Ballan. Exploiting socially-aware tasks for embodied social navigation. *arXiv preprint arXiv:2212.00767*, 2022. 8, 73
- [30] John Canny. *The complexity of robot motion planning*. MIT press, 1988. 16
- [31] Vincent Cartillier, Zhile Ren, Neha Jain, Stefan Lee, Irfan Essa, and Dhruv Batra. Semantic mapnet: Building allocentric semanticmaps and representations from egocentric views. *AAAI*, 2021. xi, 31, 32, 33, 39, 40, 41, 42, 47

- [32] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. In *Intl. Conf. on 3D Comput. Vis.*, 2017. 15, 17, 31, 32, 38, 49, 51, 60, 74, 76
- [33] Matthew Chang, Arjun Gupta, and Saurabh Gupta. Semantic visual navigation by watching youtube videos. *arXiv preprint arXiv:2006.10034*, 2020. 14
- [34] Oscar Chang, Lampros Flokas, Hod Lipson, and Michael Spranger. Assessing satnet’s ability to solve the symbol grounding problem. In *NeurIPS*, 2020. 96, 98
- [35] Devendra Singh Chaplot, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov. Learning to explore using active neural slam. In *ICLR*, 2019. 14, 31, 32, 35, 36, 38, 39, 50
- [36] Devendra Singh Chaplot, Dhiraj Prakashchand Gandhi, Abhinav Gupta, and Russ R Salakhutdinov. Object goal navigation using goal-oriented semantic exploration. In *NeurIPS*, volume 33, pages 4247–4258, 2020. 14, 16, 31, 32, 35, 36, 38, 40, 41, 42, 49, 50, 55, 74, 76
- [37] Devendra Singh Chaplot, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta. Neural topological slam for visual navigation. In *CVPR*, 2020. 31, 50, 76
- [38] Devendra Singh Chaplot, Lisa Lee, Ruslan Salakhutdinov, Devi Parikh, and Dhruv Batra. Embodied multimodal multitask learning. In *IJCAI*, pages 2442–2448, 2021. 51
- [39] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. Audio-visual embodied navigation. *environment*, 97:103, 2019. 50
- [40] Changan Chen, Yuejiang Liu, Sven Kreiss, and Alexandre Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2019. 76
- [41] Changan Chen, Unnat Jain, Carl Schissler, Sebastia Vicenc Amengual Gari, Ziad Al-Halah, Vamsi Krishna Ithapu, Philip Robinson, and Kristen Grauman. SoundSpaces: Audio-visual navigation in 3d environments. In *ECCV*, 2020. 4
- [42] Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *CVPR*, pages 12538–12547, 2019. 49

- [43] Tao Chen, Saurabh Gupta, and Abhinav Gupta. Learning exploration policies for navigation. *arXiv preprint arXiv:1903.01959*, 2019. 17, 50
- [44] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020. 47
- [45] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, pages 2172–2180, 2016. 98
- [46] Yu Fan Chen, Michael Everett, Miao Liu, and Jonathan P. How. Socially aware motion planning with deep reinforcement learning. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. 76
- [47] Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2017. 76
- [48] Francois Chollet. *Deep learning with Python*. Simon and Schuster, 2021. 5
- [49] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017. 109
- [50] Tim Crane and Sarah Patterson. *History of the mind-body problem*. Routledge, 2012. 3
- [51] Stephen Cresswell, Thomas Leo McCluskey, and Margaret Mary West. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review*, 28(2):195–213, 2013. 33
- [52] ML Cummings. The surprising brittleness of ai. *Women Corporate Directors*, 2020. 3
- [53] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *NIPS*, 2019. 98
- [54] Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep symbolic learning: Discovering symbols and rules from perceptions. *arXiv preprint arXiv:2208.11561*, 2022. 95
- [55] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI*, 2011. 97

- [56] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–10, 2018. 46
- [57] Aaron Defazio and Samy Jelassi. Adaptivity without compromise: a momentumized, adaptive, dual averaged gradient method for stochastic optimization. *Journal of Machine Learning Research*, 23:1–34, 2022. 107
- [58] Matt Deitke, Dhruv Batra, Yonatan Bisk, Tommaso Campari, Angel X Chang, Devendra Singh Chaplot, Changan Chen, Claudia Pérez D’Arpino, Kiana Ehsani, Ali Farhadi, et al. Retrospectives on the embodied ai workshop. *arXiv preprint arXiv:2210.06849*, 2022. 1, 48, 50, 57, 74, 76
- [59] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, et al. Proctor: Large-scale embodied ai using procedural generation. *arXiv preprint arXiv:2206.06994*, 2022. 47, 49, 74
- [60] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artif. Intell.*, 244:143–165, 2017. doi: 10.1016/j.artint.2015.08.011. URL <https://doi.org/10.1016/j.artint.2015.08.011>. 97
- [61] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018. 47
- [62] Kuan Fang, Alexander Toshev, Li Fei-Fei, and Silvio Savarese. Scene memory transformer for embodied agents in long-horizon tasks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 538–547, 2019. v, 13, 15, 16, 21, 30, 32
- [63] Gonzalo Ferrer, Anais Garrell, and Alberto Sanfeliu. Social-aware robot navigation in urban environments. In *Proc. of the European Conference on Mobile Robots*, 2013. 76
- [64] Zipeng Fu, Ashish Kumar, Ananye Agarwal, Haozhi Qi, Jitendra Malik, and Deepak Pathak. Coupling vision and proprioception for navigation of legged robots. In *CVPR*, pages 17273–17283, 2022. 4
- [65] Jorge Fuentes-Pacheco, José Ruiz-Ascencio, and Juan Rendón-Mancha. Visual simultaneous localization and mapping: a survey. *Artificial Intelligence Review*, 43(1), 2015. 50

- [66] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *CoRR*, abs/1609.05518, 2016. 33
- [67] Georgios Georgakis, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, and Kostas Daniilidis. Learning to Map for Active Semantic Goal Navigation. In *ICLR*, 2021. 51
- [68] Ken Goldberg. Robotics: Countering singularity sensationalism. *Nature*, 526(7573):320–321, 2015. 3
- [69] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 5
- [70] Niv Granot, Ben Feinstein, Assaf Shocher, Shai Bagon, and Michal Irani. Drop the gan: In defense of patches nearest neighbors as single image generative models. In *CVPR*, 2022. 54
- [71] Peter Gregory and Stephen Cresswell. Domain model acquisition in the presence of static relations in the LOP system. In *IJCAI*, 2016. 33
- [72] Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A. Pires, and Rémi Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018. 82
- [73] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, pages 2616–2625, 2017. 14, 36, 38, 50
- [74] Meera Hahn, Devendra Singh Chaplot, Shubham Tulsiani, Mustafa Mukadam, James M Rehg, and Abhinav Gupta. No rl, no simulation: Learning to navigate without navigating. *NeurIPS*, 2021. 48, 50
- [75] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990. 3, 96
- [76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 56
- [77] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 47
- [78] Joao F Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *CVPR*, pages 8476–8484, 2018. 47

- [79] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018. 47
- [80] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 56
- [81] Yu-Xuan Huang, Wang-Zhou Dai, Le-Wen Cai, Stephen H Muggleton, and Yuan Jiang. Fast abductive learning by similarity-based consistency optimization. In *NIPS*, 2021. 98
- [82] Unnat Jain, Luca Weihs, Eric Kolve, Mohammad Rastegari, Svetlana Lazebnik, Ali Farhadi, Alexander G Schwing, and Aniruddha Kembhavi. Two body problem: Collaborative visual task completion. In *CVPR*, 2019. 50
- [83] Jindong Jiang, Lunan Zheng, Fei Luo, and Zhijun Zhang. Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation. *arXiv preprint arXiv:1806.01054*, 2018. 31, 36, 38
- [84] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285, 1996. 33
- [85] Elia Kaufmann, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Beauty and the beast: Optimal methods meet learning for drone racing. In *ICRA*, pages 690–696. IEEE, 2019. 50
- [86] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 16, 50
- [87] Kazuhiko Kawamura, Stephen M Gordon, Palis Ratanaswasd, Erdem Erdemir, and Joseph F Hall. Implementation of cognitive control for a humanoid robot. *International Journal of Humanoid Robotics*, 5(04):547–586, 2008. 33
- [88] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Daniel D Lee, and Vijay Kumar. End-to-end navigation in unknown environments using neural networks. *arXiv preprint arXiv:1707.07385*, 2017. 50
- [89] Apoorv Khandelwal, Luca Weihs, Roozbeh Mottaghi, and Aniruddha Kembhavi. Simple but effective: Clip embeddings for embodied ai. *CVPR*, 2022. 49

- [90] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017. 15, 49, 73, 76
- [91] Iuliia Kotseruba and John K Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94, 2020. 33
- [92] Satwik Kottur, Jose M. F. Moura, Devi Parikh, Dhruv Batra, and Marcus Rohrbach. Visual coreference resolution in visual dialog using neural module networks. In *ECCV*, September 2018. 50
- [93] Jacob Krantz, Erik Wijmans, Arjun Majundar, Dhruv Batra, and Stefan Lee. Beyond the nav-graph: Vision and language navigation in continuous environments. In *ECCV*, 2020. 49, 76
- [94] Hanard Kurutach, Aviv Tamar, Ge Yang, Stuart Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *NIPS*, 2018. 33
- [95] JE Laird. Introduction to the soar cognitive architecture. Technical report, Technical Report, 2022. 33
- [96] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987. 33
- [97] Leonardo Lamanna, Alfonso Gerevini, Alessandro Saetti, Luciano Serafini, and Paolo Traverso. On-line learning of planning domains from sensor data in pal: Scaling up to large state spaces. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI-21)*, 2021. 30, 33
- [98] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 16, 32
- [99] Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects. *Algorithmic and computational robotics: new directions*, 2001. 16
- [100] 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. 97
- [101] Antonio Lieto. *Cognitive design for artificial minds*. Routledge, 2021. 33

- [102] Yen-Chen Lin, Andy Zeng, Shuran Song, Phillip Isola, and Tsung-Yi Lin. Learning to see before learning to act: Visual pre-training for manipulation. In *ICRA*, 2020. 51
- [103] Iou-Jen Liu, Raymond Yeh, and Alexander Schwing. High-Throughput Synchronous Deep RL. In *NeurIPS*, 2020. 47
- [104] Andrew J Lohn. Estimating the brittleness of ai: Safety integrity levels and the need for testing out-of-distribution performance. *arXiv preprint arXiv:2009.00802*, 2020. 3
- [105] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multiagent navigation. *IEEE Robotics and Automation Letters*, 2(2):656–663, 2017. 76
- [106] Yazhou Lu, Xiaogang Ruan, and Jing Huang. Deep reinforcement learning based on social spatial-temporal graph convolution network for crowd navigation. *Machines*, 10(8):703, 2022. 76
- [107] Haokuan Luo, Albert Yue, Zhang-Wei Hong, and Pulkit Agrawal. Stubborn: A strong baseline for indoor object navigation. *arXiv preprint arXiv:2203.07359*, 2022. xi, 50, 55, 56, 57, 64
- [108] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011. 54
- [109] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *NIPS*, 31, 2018. 97, 98, 107, 108
- [110] John McCarthy. From here to human-level ai. *Artificial Intelligence*, 171(18): 1174–1182, 2007. 3
- [111] Drew McDermott. Gofai considered harmful (and mythical), 2015. 3
- [112] Neeta Mehta. Mind-body dualism: A critique from a health perspective. *Mens sana monographs*, 9(1):202, 2011. 3
- [113] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. 16, 32
- [114] Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In *EMNLP*, 2018. 49

- [115] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015. 16, 32
- [116] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016. 16, 32
- [117] Ronja Möller, Antonino Furnari, Sebastiano Battiato, Aki Härmä, and Giovanni Maria Farinella. A survey on human-aware robot navigation. *Robotics and Autonomous Systems*, 145:103837, 2021. 76
- [118] Hans Moravec. Ripples and puddles, 2000. 3
- [119] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *UAI*, 2012. 33
- [120] Arsalan Mousavian, Alexander Toshev, Marek Fišer, Jana Košecká, Ayzaan Wahid, and James Davidson. Visual representations for semantic target driven navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8846–8852. IEEE, 2019. 16, 32
- [121] Jyothish Pari, Nur Muhammad Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*, 2021. 54
- [122] R. Parr and S. J. Russell. Reinforcement learning with hierarchies of machines. In *NIPS*, 1997. 33
- [123] Ruslan Partsey, Erik Wijmans, Naoki Yokoyama, Oles Doboševych, Dhruv Batra, and Oleksandr Maksymets. Is mapping necessary for realistic pointgoal navigation? In *CVPR*, pages 17232–17241, 2022. 2, 4, 74, 76
- [124] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *PyTorch*, 2017. 22
- [125] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 47, 50, 82
- [126] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *ICML*, 2019. 50

- [127] Annie Murphy Paul. *The extended mind: The power of thinking outside the brain*. Eamon Dolan Books, 2021. 3
- [128] Claudia Pérez-D’Arpino, Can Liu, Patrick Goebel, Roberto Martín-Martín, and Silvio Savarese. Robot navigation in constrained pedestrian environments using reinforcement learning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. 75, 76, 77
- [129] Rolf Pfeifer and Fumiya Iida. Embodied artificial intelligence: Trends and challenges. *Embodied artificial intelligence*, pages 1–26, 2004. 1, 5
- [130] Gualtiero Piccinini. The first computational theory of mind and brain: a close look at mcculloch and pitts’s “logical calculus of ideas immanent in nervous activity”. *Synthese*, 141(2):175–215, 2004. 3
- [131] Sören Pirk, Edward Lee, Xuesu Xiao, Leila Takayama, Anthony Francis, and Alexander Toshev. A protocol for validating social navigation policies. *arXiv preprint arXiv:2204.05443*, 2022. 75, 77, 79
- [132] Santhosh K Ramakrishnan, Ziad Al-Halah, and Kristen Grauman. Occupancy anticipation for efficient exploration and navigation. In *ECCV*, pages 400–418. Springer, 2020. 50
- [133] Santhosh K Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, et al. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. *arXiv preprint arXiv:2109.08238*, 2021. 74, 75, 76, 85
- [134] Santhosh K. Ramakrishnan, Dinesh Jayaraman, and Kristen Grauman. An exploration of embodied visual exploration. *IJCV*, 2021. 50
- [135] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI. In *NeurIPS Datasets and Benchmarks Track (Round 2)*, 2021. URL <https://arxiv.org/abs/2109.08238>. 48, 49, 51, 52, 57
- [136] Santhosh Kumar Ramakrishnan, Devendra Singh Chaplot, Ziad Al-Halah, Jitendra Malik, and Kristen Grauman. Poni: Potential functions for objectgoal navigation with interaction-free learning. *arXiv preprint arXiv:2201.10029*, 2022. 74, 76

- [137] Sonia Raychaudhuri, Saim Wani, Shivansh Patel, Unnat Jain, and Angel Chang. Language-aligned waypoint (law) supervision for vision-and-language navigation in continuous environments. In *EMNLP*, pages 4018–4028, 2021. 50
- [138] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *NeurIPS*, 28, 2015. 54, 62
- [139] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. *arXiv preprint arXiv:2006.13155*, 2020. 98
- [140] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *NIPS*, 2017. 98
- [141] Zhiwei Deng Karthik Narasimhan Olga Russakovsky. Evolving graphical planner: Contextual global planning for vision-and-language navigation. In *NeurIPS*, 2020. 32, 74
- [142] Malcolm R. K. Ryan. Using abstract models of behaviours to automatically generate reinforcement learning hierarchies. In *ICML*, pages 522–529, 2002. 33
- [143] Gilbert Ryle. *The concept of mind*. Routledge, 2009. 3
- [144] Manolis Savva, Angel X Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. Minos: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017. 16, 32, 34
- [145] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *ICCV*, pages 9339–9347, 2019. 15, 25, 31, 32, 38, 47, 49, 50, 58, 73, 76
- [146] Alexander Sax, Jeffrey O Zhang, Bradley Emi, Amir Zamir, Silvio Savarese, Leonidas Guibas, and Jitendra Malik. Learning to navigate using mid-level visual priors. In *Conference on Robot Learning*, pages 791–812. PMLR, 2020. 16, 22, 23, 32
- [147] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 14, 22, 47, 60

- [148] Melanie Sclar, Graham Neubig, and Yonatan Bisk. Symmetric Machine Theory of Mind. In *ICML, 2022*. URL <https://proceedings.mlr.press/v162/sclar22a.html>. 3
- [149] Luciano Serafini and Paolo Traverso. Learning abstract planning domains and mappings to real world perceptions. In *International Conference of the Italian Association for Artificial Intelligence*, pages 461–476. Springer, 2019. 30
- [150] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, and Sergey Levine. Time-contrastive networks: Self-supervised learning from video. In *ICRA, 2018*. 47
- [151] James A Sethian. Fast-marching level-set methods for three-dimensional photolithography development. In *Optical Microlithography IX*, volume 2726, pages 262–272. International Society for Optics and Photonics, 1996. 16, 37, 50
- [152] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning*, pages 3067–3075. PMLR, 2017. 110
- [153] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Claudia Pérez-D’Arpino, Shyamal Buch, Sanjana Srivastava, Lyne P Tchammi, et al. igibson 1.0, a simulation environment for interactive tasks in large realistic scenes. *IROS 2021, arXiv:2012.02924*, 2021. 4, 6, 73, 76
- [154] 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 *ICCV*, pages 2881–2890, 2019. 51
- [155] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *CVPR, 2020*. URL <https://arxiv.org/abs/1912.01734>. 4
- [156] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *CoRL*, pages 894–906. PMLR, 2022. 4
- [157] Linda Smith and Michael Gasser. The development of embodied cognition: Six lessons from babies. *Artif. Life*, 11(1–2):13–30, January 2005. ISSN 1064-5462. doi: 10.1162/1064546053278973. URL <https://doi.org/10.1162/1064546053278973>. 29

- [158] Cyrill Stachniss, John J. Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, Springer Handbooks, pages 1153–1176. Springer, 2016. 31
- [159] Luc Steels. The symbol grounding problem has been solved. so what’s next. *Symbols and embodiment: Debates on meaning and cognition*, pages 223–244, 2008. 3
- [160] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J. Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, Anton Clarkson, Mingfei Yan, Brian Budge, Yajie Yan, Xiaqing Pan, June Yon, Yuyang Zou, Kimberly Leon, Nigel Carter, Jesus Briales, Tyler Gillingham, Elias Mueggler, Luis Pesqueira, Manolis Savva, Dhruv Batra, Hauke M. Strasdat, Renzo De Nardi, Michael Goesele, Steven Lovegrove, and Richard Newcombe. The Replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 15
- [161] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 978-0-262-19398-6. URL <https://www.worldcat.org/oclc/37293240>. 96
- [162] Andrew Szot, Alex Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *arXiv preprint arXiv:2106.14405*, 2021. 47
- [163] Lei Tai, Jingwei Zhang, Ming Liu, and Wolfram Burgard. Socially compliant navigation through raw depth inputs with generative adversarial imitation learning. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 76
- [164] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *CoRR*, abs/1804.10332, 2018. URL <http://arxiv.org/abs/1804.10332>. 2, 5
- [165] Sever Topan, David Rolnick, and Xujie Si. Techniques for symbol grounding with satnet. In *NIPS*, 2021. 96, 97, 98
- [166] Peter Trautman and Andreas Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010. 76
- [167] Efthymia Tsamoura, Timothy Hospedales, and Loizos Michael. Neural-symbolic integration: A compositional perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5051–5060, 2021. 98

- [168] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008. 76
- [169] Francisco J Varela, Evan Thompson, and Eleanor Rosch. *The embodied mind, revised edition: Cognitive science and human experience*. MIT press, 2017. 3
- [170] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 14, 15, 16, 21, 32
- [171] David Vernon. *Artificial cognitive systems: A primer*. MIT Press, 2014. 33
- [172] Nam Vo, Nathan Jacobs, and James Hays. Revisiting im2gps in the deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 2621–2630, 2017. 54
- [173] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554. PMLR, 2019. 96, 98
- [174] 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 *CVPR*, 2019. 74
- [175] Xin Eric Wang, Vihan Jain, Eugene Ie, William Yang Wang, Zornitsa Kozareva, and Sujith Ravi. Environment-agnostic multitask learning for natural language grounded navigation. In *ECCV*, pages 413–430, 2020. 51
- [176] Saim Wani, Shivansh Patel, Unnat Jain, Angel Chang, and Manolis Savva. Multion: Benchmarking semantic map memory using multi-object navigation. *NeurIPS*, 33:9700–9712, 2020. vi, 47, 48, 51, 52, 53, 57, 58, 60, 61, 76
- [177] Justin Wasserman, Karmesh Yadav, Girish Chowdhary, Abhinav Gupta, and Unnat Jain. Last-mile embodied visual navigation. In *CORL*, 2022. 48, 50
- [178] Luca Weihs, Matt Deitke, Aniruddha Kembhavi, and Roozbeh Mottaghi. Visual room rearrangement. In *CVPR*, 2021. 48
- [179] Erik Wijmans, Samyak Datta, Oleksandr Maksymets, Abhishek Das, Georgia Gkioxari, Stefan Lee, Irfan Essa, Devi Parikh, and Dhruv Batra. Embodied Question Answering in Photorealistic Environments with Point Cloud Perception. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 76

- [180] 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 *ICLR*, 2019. 14, 16, 17, 23, 26, 30, 32, 47, 49, 58, 74, 76, 83
- [181] David E Wilkins. *Practical planning: extending the classical AI planning paradigm*. Elsevier, 1988. 1, 3
- [182] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In *AAAI*, 2022. xii, 98, 108
- [183] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6750–6759, 2019. 16
- [184] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari, and Yuan-dong Tian. Learning and planning with a semantic model. *arXiv preprint arXiv:1809.10842*, 2018. 16
- [185] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: real-world perception for embodied agents. In *CVPR*. IEEE, 2018. 15, 22, 32, 49, 85
- [186] Fei Xia, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchappmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese. Interactive gibbon benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020. 15, 75, 77
- [187] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, Proceedings of Machine Learning Research, pages 5498–5507, 2018. 97
- [188] Karmesh Yadav, Santhosh Kumar Ramakrishnan, John Turner, Aaron Gokaslan, Oleksandr Maksymets, Rishabh Jain, Ram Ramrakhya, Angel X Chang, Alexander Clegg, Manolis Savva, Eric Undersander, Devendra Singh Chaplot, and Dhruv Batra. Habitat challenge 2022. <https://aihabitat.org/challenge/2022/>, 2022. 5
- [189] Karmesh Yadav, Ram Ramrakhya, Arjun Majumdar, Vincent-Pierre Berges, Sachit Kuhar, Dhruv Batra, Alexei Baevski, and Oleksandr Maksymets. Offline visual representation learning for embodied navigation. *arXiv preprint arXiv:2204.13226*, 2022. 48

- [190] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997. doi: 10.1109/CIRA.1997.613851. 50, 55
- [191] F. Yang, D. Lyu, B. Liu, and S. Gustafson. PEORL: integrating symbolic planning and hierarchical reinforcement learning for robust decision-making. In *IJCAI*, 2018. 33
- [192] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. Visual semantic navigation using scene priors. *arXiv preprint arXiv:1810.06543*, 2018. 16
- [193] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *IJCAI*, 2020. 98, 108
- [194] Joel Ye, Dhruv Batra, Abhishek Das, and Erik Wijmans. Auxiliary tasks and exploration enable objectnav. *arXiv preprint arXiv:2104.04112*, 2021. 38, 49, 74, 76, 82, 93
- [195] Joel Ye, Dhruv Batra, Erik Wijmans, and Abhishek Das. Auxiliary tasks speed up learning point goal navigation. In *CORL*, 2021. 47, 49, 76, 81, 82, 84, 85, 86
- [196] Naoki Yokoyama, Qian Luo, Dhruv Batra, and Sehoon Ha. Learning Robust Agents for Visual Navigation in Dynamic Environments: The Winning Entry of iGibson Challenge 2021. *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022. 75, 77, 83, 84, 85
- [197] Wenhao Yu, Visak CV Kumar, Greg Turk, and C. Karen Liu. Sim-to-real transfer for biped locomotion. In *IROS*, pages 3503–3510, 2019. doi: 10.1109/IROS40897.2019.8968053. 2, 5
- [198] Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3712–3722, 2018. 16, 22, 31, 32, 36, 38
- [199] Fengda Zhu, Yi Zhu, Xiaojun Chang, and Xiaodan Liang. Vision-language navigation with self-supervised auxiliary reasoning tasks. In *CVPR*, 2020. 74
- [200] Fengda Zhu, Yi Zhu, Vincent Lee, Xiaodan Liang, and Xiaojun Chang. Deep learning for embodied vision navigation: A survey. *arXiv preprint arXiv:2108.04097*, 2021. 74

- [201] 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 *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017. 46
- [202] Hankz Hankui Zhuo and Qiang Yang. Action-model acquisition for planning via transfer learning. *Artif. Intell.*, 212:80–103, 2014. 33

Appendix A

Publications

This research activity has led to several publications in international conferences. These are summarized below.

A.1 International Conferences and Workshops

1. **T. Campari**, P. Eccher, L. Serafini, L. Ballan. "Exploiting scene-specific features for object goal navigation", *European Conference on Computer Vision, Glasgow, 2020, (ECCVW 2020)*
2. **T. Campari**, L. Lamanna, P. Traverso, L. Serafini, L. Ballan. "Online Learning of Reusable Abstract Models for Object Goal Navigation", *Proc. of IEEE/CVF Computer Vision and Pattern Recognition, New Orleans, 2022 (CVPR 2022)*

A.2 ArXiv Papers

1. M. Deitke, D. Batra, Y. Bisk, **T. Campari**, A. X. Chang, D. S. Chaplot, C. Chen, C. Pérez D'Arpino, K. Ehsani, A. Farhadi, Li Fei-Fei, A. Francis, C. Gan, K. Grauman, D. Hall, W. Han, U. Jain, A. Kembhavi, J. Krantz, S. Lee, C. Li, S. Majumder, O. Maksymets, R. Martín-Martín, R. Mottaghi, S. Raychaudhuri, M. Roberts, S. Savarese, M. Savva, M. Shridhar, N. Sünderhauf, A. Szot, B. Talbot, J. B. Tenenbaum, J. Thomason, A. Toshev, J. Truong, L. Weihs, J. Wu. "Retrospectives on the embodied ai workshop"

A.3 Under Submission

1. E. Cancelli*, **T. Campari***, L. Serafini, A. X. Chang, L. Ballan. "Exploiting Socially-Aware Tasks for Embodied Social Navigation". Under submission.¹
2. S. Raychaudhuri, **T. Campari**, U. Jain, M. Savva, A. X. Chang. "Modular-MON: Modular Multi-Object Navigation". Under submission.
3. A. Daniele, **T. Campari**, S. Malhotra, L. Serafini. "Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions". Under submission.

A.4 Extra

1. 3rd place at the MultiON Challenge 2021 @ Embodied AI Workshop (CVPR2021).
2. Visiting student at Simon Fraser University with A. X. Chang (2022).
3. Challenge organizer @ Embodied AI Workshop (CVPR2022)
4. Supervision of Master's students on their thesis.

¹* denotes equal contributions.