Towards a Planning-Based Neural-Symbolic Framework for Egocentric Agent Design

by

Xiaotian Liu

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
May 2022

Abstract

In the last decade, deep neural networks have given embodied agents many tools that can extract information from environmental data. However, tasks such as reasoning and long-term planning cannot be done effectively using deep neural networks. Symbolic methods, such as automated planning, are still methods of choice for embodied agents in most applications. The integration of a planning system with deep neural networks seems to be the natural next step for embodied agent design. However, many challenges from both the planning and deep learning side need to be solved for such hybrid systems. This work proposes a neural-symbolic framework for constructing embodied agents, capable of semantic navigation, that can take advantage of both neural and symbolic algorithms. The framework uses neural networks for low-level information extraction and automated planners for high-level reasoning. The main challenges our approach addresses are converting traditional pansophical planners to egocentric perspectives and finding the appropriate factored representation for environmental information discovered by such agents. The thesis has two main contributions towards building a neural-symbolic embodied agents. Our first contribution is presenting a method of converting classic pansophical planning problems to egocentric alternatives. Secondly, we propose a spatial semantic graph structure to store environmental information for autonomous object navigation.

Co-Authorship

The following publications are incorporated in this work. Xiaotian Liu is responsible for writing this thesis under the supervision of Dr.Christian Muise. The first paper was used in Chapter 4 of the thesis, and the second work is used in Chapter 5 of this thesis.

- Liu X, Paredes A, Muise C. Do You See What I See? An Egocentric View of our Pansophical Planning Problems. In31st International Conference on Automated Planning and Scheduling (p. 31).
- Liu X, Muise C. A Neural-Symbolic Approach for Object Navigation. InCVPR Embodied-AI Workshop 2021.

Acknowledgments

I would like to thank and acknowledge the people who gave me support while going through my Master's journey during these difficult times. I want to thank my supervisor, Professor Chrisitan Muise, for allowing me to study under him during the past two years. You have provided me with great research and personal support. In addition to research, you have taught me how to be a better team member and supportive of others during my time at MuLab. Without your generous suggestions and constructive criticism, I would not be able to accomplish my degree in a satisfactory manner. It was truly a pleasure to have you as a mentor and work with you in the past two years. I want to thank my mother and father for being my emotional anchor and providing unconditional love and support. Without your sacrifices, I would not be able to achieve anything on my own. I would like to thank other members of Mulab for going through this journey with me together. I want to thank Victoria Armstrong, Minhaj Ansari, and Brennan Cruse for being great team members and thank Alison Parades and Nisha Simon for providing their insights into academia. Finally, I want to thank Professor Sidney Givigi and Professor Francois Rivest for being on my thesis committee and all the Queen's School of Computing faculty members for their guidance and assistance during my time here.

Abbreviations and Symbols

CNN Convolutional Neural Network.

EP Egocentric Planner.

ESE Egocentric Subset Extractor.

IER Iterative Exploration via Replanning.

PDDL Planning Domain Definition Language.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

SLAM Simultaneous Localization And Mapping.

SLG Semantic Location Graph.

STRIPS Stanford Research Institute Problem Solver.

Contents

Abstract **i**

Co-Authorship **ii**

Acknowledgments **iii**

Contents **v**

List of Tables **vii**

List of Figures **viii**

Chapter 1: Introduction **1**
1.1 Motivation 1
1.2 Problem 3
1.2.1 Problems in Egocentric Planning 4
1.2.2 Representation Problem 5
1.3 Objective 7
1.4 Organization of Thesis 8

Chapter 2: Background **9**
2.1 Semantic Navigation 9
2.2 Planning 11
2.2.1 PDDL 11
2.2.2 Modified STRIPS Notation 11
2.2.3 Grid Navigation Example 12
2.2.4 AI2-THOR 14

Chapter 3: Methodology **16**
3.1 Semantic Segmentation Network(SSN) 17
3.2 Semantic Location Graph(SLG) 18
3.3 Egocentric Planner(EP) 19

Chapter 4: Do You See What I See? An Egocentric View of our Pansophical Planning Problems **20**
4.1 Introduction . . . 21
4.2 Approach . . . 22
4.2.1 Egocentric Subset Extractor . . . 23
4.2.2 Iterative Exploration via Replanning . . . 26
4.3 Evaluation . . . 28
4.4 Discussion . . . 30

Chapter 5: A Neural Symbolic Approach for Object Navigation **33**
5.1 Introduction . . . 34
5.2 Approach . . . 35
5.2.1 Object Detection . . . 36
5.2.2 Depth Estimation . . . 37
5.2.3 Spatial Graph Generation . . . 38
5.2.4 Egocentric Planning . . . 39
5.3 Experiments . . . 40
5.4 Results . . . 41
5.4.1 Semantic Segmentation and Depth Prediction . . . 41
5.4.2 Object Navigation . . . 44
5.4.3 Pick-and-Place . . . 45
5.5 Conclusion . . . 47

Chapter 6: Disscussion **48**
6.1 Embodied Agent Navigation . . . 48
6.1.1 Geometric based Navigation System . . . 49
6.1.2 Learning based Navigation Systems . . . 50
6.2 Planning under Partial Observability . . . 51

Chapter 7: Conclusion **54**
7.1 Summary . . . 54
7.2 Limitations . . . 56
7.3 Future Work . . . 57

List of Tables

4.1 Results of tested planning domains 30

5.1 Navigation Results 44

5.2 Summary of Object Navigation Results 45

5.3 Pick-and-Place Results 46

5.4 Failure Breakdown 46

List of Figures

2.1 Examples of Semantic Navigation [27] 10

2.2 Graphical illustration for Search-and-Rescue 12

2.3 PDDL for Search-and-Rescue 13

2.4 Examples of AI2-THOR Environments from ai2thor.allenai.org 15

3.1 An overview of the proposed framework 17

4.1 Initial condition extracted by ESE 26

4.2 An exploration action example 28

5.1 Information Extraction form Raw Image 37

5.2 Example of a Learned Spatial Memory Graph 39

5.3 Example of ALFRED dataset[42] 41

5.4 Object Recognition 43

5.5 Depth Prediction 43

Chapter 1

Introduction

1.1 Motivation

In recent years, deep learning has dramatically impacted the development of Artificial Intelligence (AI). End-to-end differentiable neural networks have achieved remarkable results in areas such as computer vision(CV), natural language processing (NLP), and reinforcement learning(RL) [52, 46, 43]. Deep neural networks can process and extract features automatically from large datasets. In contrast, symbolic AI often relies on hand-crafted rules by human experts with domain knowledge. Deep learning-based methods have come close to human-level performances in many tasks for learning unstructured data such as images and natural language [52, 46]. Deep RL systems such as AlphaGo and AlphaZero have achieved superhuman level performances in games such as Go and Chess, where we traditionally associate with logic and planning [43, 44]. One would assume deep neural networks could be suitable for building intelligent embodied agents where an agent needs to explore and carry out everyday human tasks. However, deep RL systems have rarely been used for embodied agents applications in real-life settings. One of the main reasons being deep

RL's performance deteriorates quickly when the task has sparse rewards or requires multi-step planning. Deep RL learns a policy through weights distributed in large neural networks. Therefore knowledge of the world is often entangled with rewards signals. Furthermore, the black-box nature of deep RL systems makes them difficult to interpret, thus preventing them from deploying in many common applications [14]. Deep RL for embodied agents also require billions of sample steps generated through training episodes, which makes them difficult to apply beyond virtual environments [53]. Model-based RL such as Dreamer has been used to form world models in the latent space [18]. The idea is for RL agents to learn world dynamics and plan their actions according to the learned model. However, the latent space representation entangles rewards with world dynamics without factorization, making them unsuitable for goal-oriented planning without clear rewards. Although these model-based RL agents have shown performances on par with model-free methods in benchmarks such as Atari, they have not demonstrated their superiority in solving tasks with sparse reward and long-horizon reasoning.

Symbolic methods are designed to conduct long chains of reasoning. These systems tackle reasoning by constructing explicit states and rules in the symbolic space, making them interpretable by human beings [9]. When a problem can be represented as factored symbols, long-term reasoning tasks that are difficult for deep RL systems can be trivial to solve using a generic search algorithm. However, the performance of symbolic systems is often bottlenecked by how well we can ground unstructured input into the correct factored representation.

Neural and symbolic methods appear to have distinct advantages that can complement each other's shortcomings. However, in most of their history, machine learning

and symbolic reasoning have been developed by distinct communities. With recent advancements in deep learning, more and more attention has been given to the integration of symbolic and statistical methods [40]. Building hybrid systems that can conduct symbolic reasoning with concepts acquired from unstructured data has become a promising direction for embodied agent design. There are many open questions regarding building neural symbolic agents, yet little work has been done formally to show the feasibility of such models. The cycle of perception and action requires transferring information between statistical and symbolic representations. There is also the question of what symbolic methods are suitable for such systems.

Motivated by the need for a formal framework for neural symbolic embodied agents, we propose an object-centric hybrid approach that uses automated planning techniques to reason over information grounded by neural networks. We test our approach on semantic navigation (or object navigation) tasks, one of the essential tasks in embodied agent applications. We show the advantages of hybrid systems while exploring the challenges of combining neural and symbolic methods.

1.2 Problem

To propose a framework for planning-based neural symbolic agents, we must address egocentricity in classical planning methods while finding a suitable representation to integrate planning with deep neural networks. Planning approaches work best when the world is pansophical where that state of the world is fully known and observable by an acting agent. However, in practice, embodied agents are primarily egocentric and can only observe part of the environment. Thus the challenge is finding a way to take account of egocentricity without losing the effectiveness of classic planning methods.

The other problem is using the correct level of representation learned through a neural network. There are often trade-offs between the amount of information that can be represented and the interpretability of the system. Therefore, we need to decide what level of representation is best suited for embodied agent tasks. We will discuss challenges from both areas in the next sections.

1.2.1 Problems in Egocentric Planning

Automated planning is a branch of AI that allows intelligent agents to interact with their environments to achieve certain goals. It has a long history in embodied agent design and has shown successes in applications such as service and industrial robots [25].Classical planning approaches assume the world is closed and all relevant information is known in advance. However, this is not the case in many settings where an agent has an egocentric perspective. The traditional way of dealing with uncertainties is through partially observable methods, such as contingent and conformant planning [42, 6]. Partially observable planning approaches form belief states of the world that account for multiple possibilities for the current state. Thus, they can be considered as conducting close-world planning in belief states. One of the major downsides of these approaches is that they do not scale well with increasing complexity making them too computationally expensive to deploy in real-life settings. Continual planning is another alternative class of planning approaches that solves partial observable problems. Rather than account for multiple branches of possible states combinations, continual planning uses the plan-execution-monitoring cycle to execute part of the complete plan while gathering information during execution. A principled conditional planning language, Multiagent Planning Language (MAPL),

has been proposed which hides conditional sub-plans with "unknown" fluents to guide goal-oriented exploration [8].

Although many formalisms have been proposed to solve partially observable planning problems, most egocentric agents are still implemented in an ad-hoc manner. Agent designers often prefer to implement a classic planner as sub-routines rather than modeling the entire problem as a partially observable domain. Some of the advantages of classical planning are that they are more robust, and domain designers are often more comfortable with common planning languages such as Planning Domain Definition Language (PDDL). Another challenge with partially observable formalisms is the need for the closed-world assumption where goals need to be achieved from an initial set of fluents (or their beliefs). In practice, this requires the users to define all objects in a domain prior to execution. These requirements are not realistic in many settings where exploration is required. Thus, existing partially observable formalisms are mostly used in multi-agent settings where the environment and dynamics are known, but agent behaviors are unpredictable. In egocentric agents, the world is hidden, and exploration is needed to discover information. Information of object existence and how many objects are there is often unknown to the domain designers. A framework that can complement the shortcomings of existing partially observable planning approaches is needed for egocentric agents to solve tasks in unknown environments.

1.2.2 Representation Problem

A good representation is crucial for both learning and reasoning. However, deep learning and symbolic AI seems to represent information on opposite sides of the

scale. Deep learning use gradient-based distributed representation where information is encoded as continuous vectors. Symbolic methods use localist approaches where concepts are associated with discrete symbols. Deciding on the suitable representation to bridge neural and symbolic models is necessary for any hybrid system. There are many possible approaches to integrate neural and symbolic representations. One type of approach encodes symbolic knowledge into the weights of neural networks via the network structures [41]. Another way is to extract symbolic information in continuous vectors learned through networks via disentanglement [21]. Discretized vector embeddings learned via autoencoder have also shown usefulness as state representations in classic planning tasks [1]. Most recent approaches focus on lower-level representation learned via neural networks with labeled data. One of the short-comings is that extracting symbols can only be done through a large amount of labeled relational information. These labels are not readily available and much harder to produce than simpler tasks such as classification. In addition, rules and logic learned through these networks are still distributed within large neural networks and largely uninterpretable by users. These downsides make them difficult to be applied to embodied agent applications. In most settings, embodied agents are designed for specialized tasks and are grounded in real-life objects.

Many common tasks can be modeled in a high-level symbolic system with discrete representations. For example, modeling semantic navigation tasks only requires an agent to understand the concept of objects and their spatial relationships. The benefit of learning comes from converting raw sensory data and their uncertainties into structured symbols. Grounding unstructured information often means conducting classification, which is one of the most applied areas for deep learning. However, the

output for a classification problem is a distribution over categorical labels. The relationship among common output classes is often not incorporated in the classification task. Thus, an intermediate representation is still required to bridge neural network outputs with the information necessary for planning. One of the main challenges of a neural symbolic model is finding a way to bridge neural network outputs and their uncertainties with inputs for symbolic systems.

1.3 Objective

The main objective of this work is to propose a planning-based neural symbolic framework for embodied agents that can conduct semantic navigation tasks. To achieve our objective, the proposed framework must address two problems. The first is to incorporate egocentricity into classical planning methods where exploration and information gathering are required. In addition, the solution should be able to convert classical planning methods into egocentric alternatives semi-automatically for ease of use. Secondly, the proposed framework needs to have a representation that can bridge the output distribution of neural networks with inputs required for symbolic planning. The approach needs to demonstrate effectiveness in semantic navigation tasks where the goal is to discover a predefined set of objects in an unknown environment. Our work should be compared with existing end-to-end deep learning methods based on accuracy and sample efficiency. Overall, the thesis highlight the feasibilities and challenges of building planning-based neural symbolic agents. Our work to serve as a proof of concept for the effectiveness of hybrid systems and as a foundation for future work in neural symbolic embodied agent design.

1.4 Organization of Thesis

Chapter 2 introduces the relevant background information for semantic navigation, egocentric planning, and object detection. Chapter 3 gives an overview of the proposed neural symbolic framework and its main components. Chapter 4 details an open-loop replanning method that can convert pansophical planning problems to egocentric alternatives. The method is tested on classical planning problems commonly used in planning benchmarks. Chapter 5 shows the integration of object detection with egocentric planning that can conduct semantic navigation tasks. The advantages of the hybrid method versus the end-to-end differentiable model are also outlined in the chapter. Chapter 6 outlines related work in both agent navigation and partial observable planning. Finally, Chapter 7 summarizes our work and outlines the thesis's feasibility, limitation, and future directions.

Chapter 2

Background

2.1 Semantic Navigation

We use semantic navigation as the main evaluation methods for our proposed method. Semantic navigation (or object navigation) is one of the most critical problems to solve in embodied agents. It refers to to the task of gathering information about the world and converting it into semantic meaning. Human beings possess the natural ability to conduct semantic navigation. Thus, we can easily reason about the world around us and perform complex tasks accordingly. Embodied agents are designed to perform tasks in many areas such as space exploration, transportation, and security. However, most of the embodied agents today do not possess the ability to understand and reason over the environment. With the advancements in sensor technologies and computational power, integrating semantic information has become a more relevant topic task for embodied agent design. We work aims to construct agents that can eventually reason like humans on both geometric and semantic levels.

Spatial and conceptual information are both required for semantic navigation. Embodied agents and human beings store information differently. The inner working

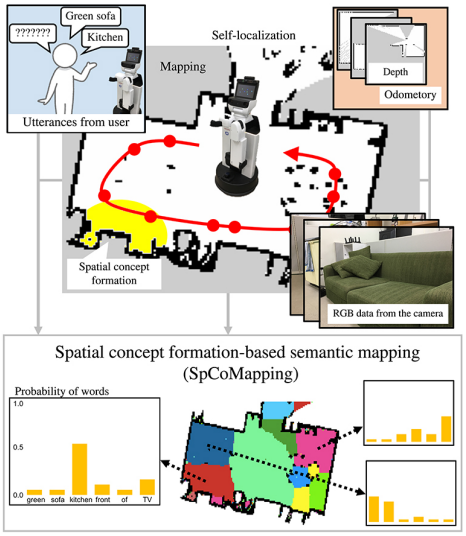

Figure 2.1: Examples of Semantic Navigation [27]

of the human brain remains mostly a mystery. However, the representation of both spatial and semantic information seems to be entangled in the brain [56]. Most embodied agent designs do not aim to replicate the same system as the human brain. Rather, embodied agents are designed to mimic the behavior of human beings. In most robots, location information is stored in a tubular coordinate system. Semantic information is stored separately to be used to reason over the space. The key to semantic navigation is to build a system that can bridge the gap between spatial and semantic information. Such a system is often referred to as a semantic mapping [27]. A example of semantic navigation system is shown in Figure 2.1 In this work, we use a graph representation of possible movements as a way to store semantic and spatial information to bridge this gap.

2.2 Planning

2.2.1 PDDL

Planning Domain Definition Language (PDDL) was initially developed by Drew McDermott and then subsequently improved upon by the automated planning community to standardize domain and problem definitions [32]. It is based on Stanford Research Institute Problem Solver, or STRIPS, a formalism where states and actions are described by binary literals of atoms. Planning tasks described in PDDL consist of domain descriptions, initial conditional of the state, and the goal state. The domain descriptions consist of objects, predicates, and actions relevant to the planning problem. The initial condition and goal condition describe both the current and goal state in the form of grounded fluents. Additional constructs such as duration and action cost are added to subsequent versions of PDDL to incorporate more problem types [15]. We use PDDL to represent planning problems in this work.

2.2.2 Modified STRIPS Notation

We use an extension the commonly used STRIPS notation to include the notion of objects. In STRIPS, a planning problem is defined with a tuple $\langle F, I, A, G \rangle$. F is a set of fluents, $I \subseteq F$ is the initial state of the world, and $G \subseteq F$ is the goal state. A is the set of actions available. For each action $a \subseteq A, PRE(a) \subseteq F$ is the precondition of that action, $ADD(a) \subseteq F$ is the add effect, and $DEL(a) \subseteq F$ is the delete effect. Our method operates over planning problems specified in PDDL. Therefore, we consider fluents to be represented by predicates with typed objects. We define a set O to be the set of objects in each planning problem. The explicitly list of objects in our modified notation is to help our algorithms to identify objects and their types

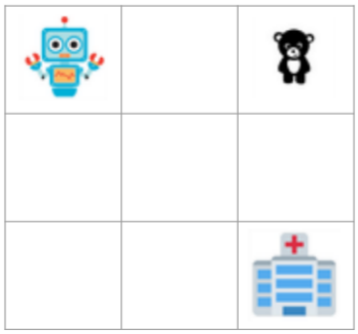

Figure 2.2: Graphical illustration for Search-and-Rescue

when new objects are encountered during exploration. We include O in the STRIPS problem, and our complete planning problem is defined as a tuple $P = \langle O, F, I, A, G \rangle$. We use $\textsc{Progress}(I, a) = I'$ to indicate progression when an action a is taken and the initial condition I is to $'I$ after the action. The operator $\textsc{Solve}(P)$ is used when solving the planning problem P with an planner. This modified STRIPS notation is ued in our egocentric planning in Chapter 4.

2.2.3 Grid Navigation Example

We use a 2D navigation problem specified in PDDL as an example to illustrate our approaches throughout this paper. The problem is the simplified version of the Search-and-Rescue problem [49]. In this planning problem, an agent is spawned on a 2D grid, and its goal is to search and pick up a person and then navigate to the hospital. The planning problem includes 4 object types: a robot object, location objects, a person object, and a hospital object. The predicates include object location predicates, connection predicates connecting locations, and a holding predicate indicating whether an agent is holding a person. The action set includes moving to another location and picking up the person. A visual illustration of the problem with the corresponding

PDDL definitions are shown in Figures 2.2 and 2.3.

```
DOMAIN:
(define (domain searchandrescue)
  ......
  (:predicates
    (conn ?v0 - location ?v1 - location ?v2 - direction)
    (robot-at ?v0 - robot ?v1 - location)
    (person-at ?v0 - person ?v1 - location)
    (hospital-at ?v0 - hospital ?v1 - location)
    (carrying ?v0 - robot ?v1 - person)
    (handsfree ?v0 - robot)
    (move ?v0 - direction)
    (pickup ?v0 - person)
    (dropoff))

    (:action move-robot
        :parameters (?robot - robot ?from - location ?to - location ?dir - direction)
        :precondition (and (move ?dir)
            (conn ?from ?to ?dir)
            (robot-at ?robot ?from))
        :effect (and
            (not (robot-at ?robot ?from))
            (robot-at ?robot ?to))  )

    (:action pickup-person ......)

    (:action dropoff-person ......) ) )

PROBLEM:
(define (problem searchandrescue)
    (:domain searchandrescue)
    ......
    (:init
        (conn f0-0f f0-1f right)
    ......
        (conn f2-2f f1-2f up)
        (dropoff )
        (handsfree robot0)
        (move up)
        (move down)
        (move left)
        (move right)
        (pickup person0)
        (robot-at robot0 f0-0f)
        (person-at person0 f0-2f)
        (hospital-at hospital0 f2-2f)
    (:goal
        (person-at person0 f2-2f)))
```

Figure 2.3: PDDL for Search-and-Rescue

2.2.4 AI2-THOR

This thesis uses scenes and images generated with the AI2-THOR simulator to conduct our experiments [29]. AI2-THOR uses 3D reconstruction of photo-realistic home environments, including bathrooms, bedrooms, kitchen, and living room scenes. The environment provides an egocentric agent that can interact with common household objects and perform navigation actions with simulated physics. The goal of the simulator is to provide realistic scenarios for an agent to conduct learning and planning. The simulator provides the agent with a first-person view that can be rendered as an RGB image. The object information is given as a dictionary consisting of bounding boxes and binary masks. The distance is rendered as an inverted pixel-level depth map. Due to high reconfigurability and flexibility, AI2-THOR is one of the most used environments for embodied agent design. Many types of research in reinforcement learning, visual question answering, object detection, imitation learning researches have used AI2-THOR as a way to evaluate model performances citeshridhar19. We also use the ALFRED challenge dataset provided by AI2-THOR to evaluate the long horizon sequential decision performance of our methods [42]. The ALFRED dataset contains videos of egocentric views of agents' performing common household such as heating or washing a dish in a kitchen environment. For each video frame, both depth and object information can be extracted from the AI2-THOR simulator. Human-labeled natural language descriptions are also given as high-level instructions for the agent. For our purposes, we ignore the nature language portion of the dataset and only object and goal information provided. In our experiments, we focus on deterministic and stochastic actions. We set the rotation angle to a fixed 90 and movement distance to 0.5 meters. Our agents can interact with objects within a 1.5

Figure 2.4: Examples of AI2-THOR Environments from ai2thor.allenai.org

meter radius, and objects beyond 5 meters are considered unobservable. Our agents can interact with objects with toggle, open, and close actions. Although the global controller can access states of objects, our agents can only extract information from its sensory inputs. Examples of visual inputs are shown in Figure 2.4.

Chapter 3

Methodology

Our framework consists of three modules: Semantic Segmentation Network(SSN), Semantic Location Graph(SLG), and Egocentric Planner(EP). The SSN module contains neural networks pre-trained on image segmentation tasks that can identify objects and regions of interest from an egocentric image. The SLG module is a learned graph that stores locational and observational information. The EP module is an egocentric version of classic planners. For each step, the agent parses raw image inputs using SSN to ground object and relational information. The output of SSN is stored in the nodes of SLG and is updated after every action. The semantic graph is then fed into a planner parser and is translated as state inputs for EP. Finally, EP determines whether a plan can be produced or more exploration is needed to gather additional information. When a plan can be found, the agent will use policy produced by the planner, otherwise our methods trigger an exploration policy where the goal is to gather new information required for the original goal. The overall framework can be seen in Figure 3.1, and each module will be discussed in the following sections. An implementation of our framework is tested on object navigation tasks. The results are shown in Chapter 5 of this thesis.

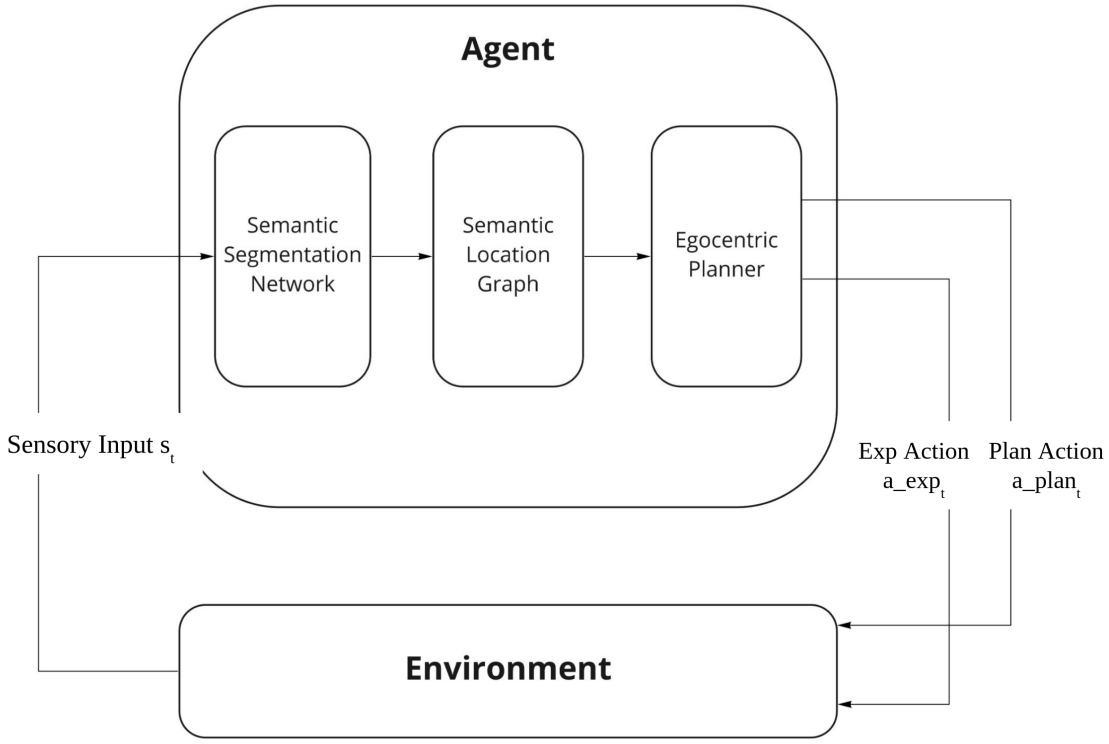

Figure 3.1: An overview of the proposed framework

3.1 Semantic Segmentation Network(SSN)

The Semantic Segmentation Network grounds the egocentric image into a structure set of objects and relationships. The exact implementation of such networks should be decided based on task and environment. Object detection and segmentation is a well-researched area. Some of the most common techniques are YOLO and MaskRCNN [38, 20]. Deep learning based scene understanding techniques, such as the Deep Relational Network, can be applied to extract relationships among objects [12]. For automated planning methods, objects and relationships are the building block for most problems. Thus it is natural for a domain designer to specify what objects and their relationship are important for a certain problem. These specifications can then

be used as categories to train supervised deep neural networks. Modularizing SSN also gives our framework better knowledge transfer capabilities than an end-to-end system. A network trained to recognize an object can be used in other tasks that involve the manipulation of that object.

3.2 Semantic Location Graph(SLG)

The Semantic Location Graph is a graphical representation that contains information produced by the Semantic Segmentation Module. Integrating semantic and locational representation of space is one of the foundational problems in embodied agent design. An egocentric agent only has immediate access to its local space where action can be taken within perception. To conduct tasks in a partially observable environment, an agent needs to find a way to understand environmental structures through exploration and experiences. Representing space as a top-down map is one of the most common approaches to represent the global space. Techniques such as SLAM have a long history in robotics and navigation[33]. However, incorporating semantic information into a top-down map requires transferring visual perception into 3D mashes with semantic labels. The translation drastically increases space complexity and is prone to errors, thus making them unsuitable for symbolic algorithms. Our framework converts semantic information into affordances, defined as locations where action can be conducted [16]. We can then represent the semantic information of an environment as a graph structure where affordances are nodes connected by navigational edges. This way, we can reduce a planning problem's complexity by sperate navigation from planning objectives. However, traditional affordances are identified based on applicable actions, which can be difficult to determine prior to action execution. Thus we

use an object-centric approach that labels a place as an affordance if a predefined set of objects can be found within perception. This way, we can reduce the task of determining action applicability to object recognition which is a much simpler problem. Another advantage of focusing on objects is to encourage knowledge transfer among different tasks. New sets of actions can easily be added to a problem during exploration if they operate on the same set of objects.

3.3 Egocentric Planner(EP)

To convert information stored in the Semantic Location Graph as inputs for a planning problem, we need to find a way to incorporate egocentricity into the planning formalism. Classical planners are defined with the fully observable assumption where the entire state space is known to the agent. However, exploring the whole state space is often impractical, and much of the environment could be irrelevant to the goal. Thus we propose an Egocentric Planner(EP) that can semi-automatically convert classical planning problems into egocentric alternatives. The method assumes that the world is made up of objects, and predicates define states and relationships among objects. Since EP could share the same object set as the SSN module, the object-centric view enables direct information transfer among modules. Furthermore, our method uses an open-loop approach which interleaves planning and exploration as separate policies. If our planner has enough information to formulate a plan, then our agent will use the actions generated from EP to achieve its goal. Otherwise, the planner will reply on an exploration policy to gather more information about the environment. The detailed implementation of our Egocentric Planner is shown in Chapter 4 of this thesis.

Chapter 4

Do You See What I See? An Egocentric View of our Pansophical Planning Problems

This work aims to propose a semi-automated mechanism that allows planning domain designers to convert classical planning problems into an egocentric alternative. Classical planning takes a pansophical view of the world: everything is fully known, observed, and static (no exogenous events with a single agent). While there are extensions to partial observability, this leapfrogs an important intermediate step of embodied agent design: egocentricity. The generated planning problems are classical as well, and we introduce an open-loop replanning mechanism that progressively explores the egocentric space until the original goal is solved (or deemed unsolvable). Our work serves as a crucial first step towards embodied agents that can be equipped with an appropriately specified egocentric version of known environment dynamics.

4.1 Introduction

Most classical planning problems are defined with a pansophical reference frame where the environment is fully observable. However, in many planning tasks, an agent only has a limited view of the environment. Existing approaches on partially observable planning problems, such as conformant or contingent planning, use belief states to represent uncertainty in the environment [22, 23]. However, representing belief states often requires access to information such as what objects are present in the environment in advance. Many planning applications do not provide access to such information. These types of egocentric agents often need to incorporate exploration in the strategy [24]. Egocentricity can also be necessary for embedded agents to transfer skills across environments [10]. Often, solving a planning problem egocentrically requires manual conversion by a domain expert. The conversion process can be time-consuming, sometimes requiring the definition of new syntax. A standardized semi-automatic approach can lift the burden of manual conversion, decreasing the hurdle of egocentric planning research and open up the door to domain-independent planning techniques to this setting.

We propose a novel semi-automatic approach that can convert classical planning problems defined in PDDL into egocentric alternatives. Our method takes a set of user-defined object types with their initial states as input, and outputs the egocentric version of the original problem. We assume that the world is made up of objects. Thus predicates define all objects' states and relationships. By defining which objects can be observed by an egocentric agent, we can extract the observable state space from the corresponding predicates. We use an exploration algorithm that searches through the original problem's state space until a plan is found or deemed impossible.

The iterative exploration is facilitated by an open-loop replanning mechanism, which progressively updates the observable state space. Thus, our overall framework consists of two interacting processes: one determines which set of fluents are observable, and the other explores the unobserved state spaces.

The egocentric planning sub-problems generated by our approach can be solved using off-the-shelf planners. Our algorithm also keeps syntax and vocabularies consistent with the original planning problem, making results easily interpretable. We test the validity of our approach by using it on five classical planning problems and corresponding egocentric interpretations of them. Our experimental results demonstrate that our approach is practical through both quantitative and qualitative evaluations. To the best of our knowledge, there has not been any attempt to semi-automatically convert classic pansophical planning problems into egocentric alternatives. More complex or domain-specific problems may require further modification of our proposed approach. Nevertheless, we believe this work is an important step forward that can encourage more automated planning research in egocentric settings.

In the following sections, we provide preliminary definitions and a detailed outline of our framework, followed by descriptions of our experimental setup and a discussion of the results. In addition, we provide a step by step example of our approach using a simple grid based planning problem.

4.2 Approach

For a planning problem to be egocentric, only parts of the state space are observable. We can construct an egocentric planning problem $P' = \langle O', F', I', A', G' \rangle$ from the original pansophical planning problem by finding the corresponding subsets for each

element in P. $\langle O', F', I' \rangle$ are the egocentric version of the problem's objects, fluents and initial states. Their value depends on an agent's observable state space. A' is the set of actions available to the egocentric agent containing O'. It contains actions in the original pansophical set A with additional actions required for exploration. G' is the goal of the egocentric agent, which is to gather information until the original goal G can be achieved. The objectives of our approach are to determine the current egocentric state and to facilitate exploration. We separate these tasks into two separate algorithms. The first algorithm extracts the observable states from an pansophical planning state space. We call this algorithm the Egocentric Subset Extractor, or ESE. The second algorithm facilitates exploration and information gathering in an partially observable environment. We call this component the Iterative Exploration via Replanning, or IER. The subsequent sections will motivate and describe both algorithms in detail, followed by a demonstrated example of our approach on the grid world domain.

4.2.1 Egocentric Subset Extractor

The ESE algorithm determines which fluents in the pansophical planning problem are observable. We adopt an object-centric view of the world and assume objects' states/relationships are represented by fluents in the form of object typed predicates. Thus, if we can determine which set of objects are observable by an agent, we can find the corresponding set of fluents that are also observable. The visibility of certain objects depends on the agent's egocentric state, and we refer to these as *anchor objects*. The ESE algorithm determines which subset of fluents in the initial set in a planning problem $P = \langle O, F, I, A, G \rangle$ should be included in the egocentric problem.

Our approach requires the user to determine the type and the initial set of observable objects. We define this set S as a tuple $\langle T, C, R \rangle$. T contains the anchoring object types, and C is a set of anchor objects currently observable. R is defined as a set of predicates that connect one anchor object to another. We make the following four assumptions for our approach:

1. We can uniquely determine a planning problem's egocentric aspects by way of the observable anchor objects.

2. If an object is observable, then its states and relationships with other objects, in the form of predicates, are also observable.

3. Any predicates that do not contain any anchor objects are always observable.

4. There exist a set of predicates that define connections or relations among anchor objects.

5. The environment is deterministic.

The first assumption is a direct result of adopting an object-centric view of the world. The second assumption allows us to determine which fluents in the form of predicates are observable. Intuitively, predicates represent the state and relationship among objects. Thus any predicates that contain observed objects should also be observable. For the third assumption, if a predicate contains no anchor objects, its observability is independent of an agent's egocentric state. We assume these predicates should always be observable by the agent. Finally, we assume anchor objects are connected via relational predicates. Agents should be able to observe other anchor objects that are immediately related to the current set of anchor objects.

Algorithm 1: Egocentric Subset Extractor

Input: $P = \langle O, F, I, A, G \rangle$
Anchor Object Set $S = \langle T, C, R \rangle$
Output: Egocentric projection $P^{'}$

Initialize $O^{'} = \emptyset$ and $I^{'} = \emptyset$;
for $p \in I$ ***where*** *type(p)* $\in R$ **do**
    if p *has object* $o \in C$ **then**
        for $o \in p$ **do**
            if *type(o)* $\in T$ **then**
                $O^{'} = O^{'} \cup \{o\}$;
for $p \in I$ ***where*** $predicate_type(p) \notin R$ **do**
    if p *has* $o \in O^{'}$ ***or*** p *is constant* **then**
        $I^{'} = I^{'} \cup \{p\}$;
        for $o \in p$ **do**
            $O^{'} = O^{'} \cup \{o\}$;
$F^{'} = F$;
$A^{'} = A$;
$G^{'} = G$;
return $P^{'} = \langle O^{'}, F^{'}, I^{'}, A^{'}, G^{'} \rangle$;

These related anchor objects are used to determine where the agent need to explore, which we will discuss in a later section. Our ESE algorithm first iterates through all the relational predicates of type R to extract observable relational predicates and their anchor objects. It then extracts all the non-relational predicates that are either always observable or containing observable anchor objects. The extracted predicates define the egocentric model of the agent. The full ESE algorithm is shown in Algorithm 1.

In the grid example, the user needs to define a set of initial observable anchor objects, anchor types and relational predicates, $S = \langle T, C, R \rangle$. Anchor objects in this problem are the location objects. The relational predicates are `(conn )`. Initially, only `f0-0f` is observable by the agent. The ESE algorithm first extracts all

```
(:init
    (conn f0-0f f0-1f right)
    (conn f0-0f f1-0f down)
    (conn f0-1f f0-0f left)
    (conn f1-0f f0-0f up)
    (dropoff )
    (handsfree robot0)
    (move up)
    (move down)
    (move left)
    (move right)
    (pickup person0)
    (robot-at robot0 f0-0f)
    )
```

Figure 4.1: Initial condition extracted by ESE

the relational predicates containing `f0-0f`. The algorithm then finds all location objects related to the `f0-0f` which are `f0-1f` and `f1-0f`. Next, the ESE algorithm extracts non-relational predicates in initial state I that contains `f0-0f`, `f1-0f`, `f0-1f`. `(robot-at robot0 f0-0f)` is extracted as a result. Finally, all fluents contain no location objects are considered observable and are also extracted. The egocentric initial state $I^{'}$ is shown in Figure 4.1.

4.2.2 Iterative Exploration via Replanning

An egocentric agent can only observe a subset of the whole state space. Thus, a way to explore an unknown environment is often required to formulate a plan. An exploration strategy must identify which part of the state space needs to be explored while keeping track of state information observed previously. Our Iterative Exploration via Replanning (IER) algorithm is such an approach that progressively explores unknown or partially known environments. We designed IER to generate and modify existing PDDL files directly. As a result, we can use off-the-shelf planners directly.

In addition to $P' = \langle O', F', I', A', G' \rangle$ generated by the ESE algorithm, IER requires the user to identify an additional set of *exploration actions*, E, as input. We define exploration actions as actions that an agent needs to change its current observed state space.

> *By taking actions in E, an agent will transition to a state where new objects and fluents are observable.*

An exploration action a must have at least one anchoring object in its parameters. Both $PRE(a)$ and $ADD(a) \cup DEL(a)$ must also have predicates containing these anchoring objects. To distinguish between observed and unobserved objects, we create `(unknown ?obj)` predicates to identify unobserved objects. For an exploration action a, `(unknown o)` predicates are added to $PRE(a)$ for each anchor object, `o`, declared in the parameters set. After the action is taken, the unknown object is deemed to be revealed and will be removed using $DEL(a)$. An example of exploration action is the *move* action in search-and-rescue domain where the agent can observe anchor object *location* when the action is taken.

When a plan with the original goal cannot be found, our algorithm sets exploration as a goal. We achieve this by introducing an `(exploration)` predicate to replace the original goal state. Since exploration is achieved through exploration actions in E, we need to add the `(exploration)` goal predicate to $ADD(a)$ for each $a \in E$. After an exploration step, a new planning problem P' is generated using ESE. These steps are repeated iteratively through replanning until a plan for the original goal can be found. IER is shown in detail in Algorithm 2.

In the grid example, the action required for the agent to move to another egocentric state is the `move-robot` action. IER first iterates through all the fluents in the

```
(:action exploration
    :parameters (?robot - robot ?from - location ?to
                 - location ?dir - direction)
    :precondition (and (move ?dir)
        (conn ?from ?to ?dir)
        (robot-at ?robot ?from)
        (unkown ?to))
    :effect (and
        (not (robot-at ?robot ?from))
        (robot-at ?robot ?to))
        (not (unkown ?to))
        (explored))
```

Figure 4.2: An exploration action example

initial condition set $I^{'}$ and identifies unknown locations by adding `(unknown f1-0f)` and `(unknown f0-1f)` fluents. To make an exploration action, our IER algorithm creates a new action `explore` by adding `(unknown location)` in `move-robot`'s precondition, `(not (unknown location))` and `(explored)` predicates in its effect set. The `exploration` action is illustrated in Figure 4.2. Since no plan can be found for $G^{'}$, we replace the goal state with $G^{'} = \{$`(explored)`$\}$. This forces the agent to move to another location to gather more information about the broader environment. In the next iteration, the ESE algorithm will extract a new set of observable objects followed by IER until a plan with the original goal can be found or the problem is deemed unsolvable.

4.3 Evaluation

We tested our approach empirically on seven classical planning domains written in PDDL. These domains are Blocks World, Minecraft, Search-and-Rescue, Sokoban, and Elevators. In addition, we used classical planning problems defined in the PDDL-Gym libarary [45], which contains domain and problem files written in PDDL, along with visualization APIs. For each planning problem, the user only needs to add the

`(unknown ?obj)` predicate, the `(explored )` predicate, and the exploration actions. No modifications are needed for the problem file. It takes only a matter of minutes for manual conversion if the user is already familiar with the domain and PDDL modelling in general.

For each planning domain, we tested eight different problem setups. The results are summarized in Table 4.1. The success rate is number of successful conversion for each domain. The egocentric and pansophical plan length is also shown. The set of exploration action and anchor objects we defined is also shown the figure. We evaluate the performance based on the percentage of successful conversions from an pansophical planning problem to the egocentric alternative. In addition, we compared the average number of steps required for the egocentric agent to solve a planning problem. We used Tarski [36] to parse each planning domain, and the actual planning is done on the Planning.Domains online solver [34]: `http://solver.planning.domains`. We also show the set of all anchor objects and exploration actions used in our experiments. Although most problems only require singular anchor object and exploration action, the Travel and Logistics domain shows that our algorithm also works when multiple anchor objects or exploration actions are required in a domain.

The results show that our method can successfully convert most classical planning problems to an egocentric version. For Search-and-Rescue, Blocks World, Elevator, Ferry, Travel and Logistics domains our algorithm can successfully convert 100% of the testing problems. In the case of Sokoban, we were able to covert four out of five problems settings. Unlike the other six problems, Sokoban contains irreversible actions that can result in dead-ends. The failure occurred when the agent can no longer achieve its original goal due to such actions. For the logistics domain, we

have to make changes to the original problem by adding local connections within a city. The original problem does not provide connections among airports and stations, thus ignores the iterative process for the agent's movement. Our method relies on connection predicates to facilitate gradual exploration. If an agent can reach a state without passing through any other state, our method will treat both states as visible in the egocentric formulation. However by simply adding geometry among locations, our algorithm is able to conduct egocentric conversion without any other problems.

	Success	**Plan Length**		**Anchor**	**Explore**
	Rate	**Ego.**	**Pan.**	**Objects**	**Actions**
Search-&-Resc.	100%	26	10	location	move
Blocks World	100%	16	11	block	pick-up
Elevator	100%	29	22	floor	move
Sokoban	75%	64	41	location	move-dir
Ferry	100%	23	12	location	move
Travel	100%	17	15	state	walk, fly
Logistics	100%	21	17	city, location	drive, fly

Table 4.1: Results of tested planning domains

4.4 Discussion

Our algorithm can successfully convert most of the domains to egocentric equivalents. This demonstrates that our approach is not limited to just agent navigation problems. Both Search-and-Rescue and Sokoban are 2D navigation problems where the notions of agent sand egocentricity are apparent. In both problems, agents are defined

explicitly by the domain designer as `agent` or `robot` objects. However, an explicit definition may not always be necessary. We can define agents as the set of actions that can change the observable state. For example, in Blocks World this action set is `{stack, unstack}`. Intuitively, the agent is whomever that have the ability to move the blocks.

The location object is an obvious choice to define egocentricity in 2D navigation problems. However, defining it for Blocks World is more challenging. The set of observable states changes depending on the perspective of the observer. Our solution adopts a (literal) top-down perspective, limiting the observable state to only blocks that are on the top of stacks. We define the block object as the anchor object. Each block is connected via `(on ?block1 ?block2)` connection predicates. We then set `unstack` action as the exploration action. A block is considered explored when the `unstack` is conducted. Our IER algorithm then extracts a new block to explore using the `(unknown ?block)` predicate. The Blocks World formulation shows as long as all required inputs can be satisfied, our method can convert a planning domain without an explicit definition of an agent.

Our method treats exploration as a way to gather information about the state-space independent of the original goal. Thus, such exploration could lead the agent to reach a dead-end where the original goal is no longer reachable. For problems like Search-and-Rescue and Blocks World, all actions are reversible, and the initial goal will always be solvable when enough information is gathered. However, in Sokoban, the agent can reach a dead-end via irreversible actions. In our implementation, our agent does not consider the feasibility of the original goal when exploring, which can be an extension of this work.

Algorithm 2: Iterative Exploration via Replanning

Input: Problem $P = \langle O, F, I, A, G \rangle$
Anchor object set $S = \langle T, C, R \rangle$
Exploration action set E
Output: Plan M for the pansophical environment

Initialize $O_e, F_e, I_e, A_e, G_e = \emptyset$;
plan $M = [\,]$;
while SOLVE$(P) = NULL$ **do**
  $O_e = O$;
  $A_e = A$;
  for $o \in O$ **do**
    if *type*$(o) \in T$ *and* $o \notin C$ **then**
      $I_e = I_e \cup \{$ `(unknown o)` $\}$;
      for $a \in E$ **do**
        $a^{'} = a.copy()$;
        $PRE(a^{'}) = PRE(a^{'}) \cup$ `(unknown o)`
        $DEL(a^{'}) = DEL(a^{'}) \cup$ `(unknown o)`
        $ADD(a^{'}) = ADD(a^{'}) \cup$ `(explored)`
        $A_e = A_e \cup a'$
  $G_e = \{$`(explored)`$\}$;
  $F_e = F \cup I_e \cup G_e$;
  $\pi =$ SOLVE$(\langle O_e, F_e, I_e, A_e, G_e \rangle)$;
  $M.extend(\pi)$;
  for $a \in \pi$ **do**
    if $a \in E$ **then**
      add anchor objects in a to C;
  $I_e =$ PROGRESS(I_e, π) ;
  $G_e = G$;
  $P =$ ESE$(\langle O_e, F_e, I_e, A_e, G_e \rangle, \langle T, C, R \rangle)$;
  where new egocentric problem P *is generated by* ESE;
$M.extend($SOLVE$(P))$;
return M;

Chapter 5

A Neural Symbolic Approach for Object Navigation

In this work, we propose a simple neural-symbolic approach for object navigation in the AI2-THOR environment. Our method takes raw RGB images as input and uses a spatial memory graph to store object, depth, and location information. By having a discrete graph representation of the environment, the agent can directly use search algorithms to find a path to the goal. For more complex problems, our method also can convert a learned graph into an automated planning formulation. An off-the-shelf planner can solve the planning problem for the agent. Overall, our architecture consists of a semantic segmentation network, a depth prediction network, a spatial graph representing the environment, and a planning converter. We test our approach for both simple object navigation tasks, which require navigating towards a target object, and a set of more complex pick-and-place tasks, which involves finding a target object and moving it to a receptacle object. Model performance is evaluated on both task completion rate and steps required to reach target objects in an unknown environment. Empirical results demonstrate that our approach can achieve near-optimal

performance for object navigation with the ability to solve long horizon sequential decision problems. Our work builds a foundation for a neural-symbolic approach that can reason via unstructured visual cues.

5.1 Introduction

Object navigation refers to the task of finding specific scenes or objects in an unknown environment [11]. Solving the object navigation problem is necessary for building intelligent systems that can autonomously conduct tasks in any given environment. Object navigation is a natural ability possessed most animals, but it is not a trivial problem for artificial agents to solve [2]. The challenges of object navigation include achieving sub-goals, such as object recognition [11] and scene memorization [11]. An agent needs to navigate, discover objects, and memorize scenes while keeping track of its location. Recent advances in deep reinforcement learning have given rise to end-to-end RL agents that can conduct semantic navigation tasks using raw RGB images [59]. However, these end-to-end systems require large neural networks with a significant amount of training steps. Additionally, the policies learned by these systems are distributed among its weights, which make transfer learning difficult [58]. In contrast, symbolic approaches, such as automated planning, can easily solve most object navigation tasks given a discrete environment [42]. The problem with symbolic approaches is that they requirement explicit defined symbols to represent environmental information.

We approach object navigation with a neural-symbolic method, taking raw RGB images as input and constructing a spatial memory graph containing both location and object information. The learned graph can be searched using any graph search

algorithm or be converted into a planning problem. Rather than embedding reward signals inside a learned policy, we use neural networks to ground visual input into discretized symbols. Our methods modularize and disentangle reasoning with perception, which should enable more sample-efficient learning. Our approach first converts raw RGB images into a semantic segmentation and a depth perception map. The training data are generated using the AI2-THOR simulator, which provides both object and depth information with an egocentric view. We chose YOLO for semantic segmentation and U-Net for depth prediction [39, 37]. The environmental information extracted is then stored based on the location and orientation of the agent in the form of a directed graph. Finally, we convert the learned graph into a PDDL problem with a predefined goal. The PDDL problem is passed to our iterative egocentric planner, shown in Chapter 4.2 of this thesis, to generate action sequences. We demonstrate that our neural-symbolic approach is very sample efficient, and the object navigation is almost trivial once the graph is constructed after some exploration. This work also builds the foundation for applying neural-symbolic methods to complex long-horizon tasks for embodied agent reasoning.

5.2 Approach

To tackle the object navigation problem, our agent needs both a visual perception module and a memory unit. The proposed method has four major components: a semantic segmentation network, a depth prediction network, a spatial graph constructor, and a module to convert the learned graph to a planning problem. We chose to use YOLO and MaskRCNN [37] as our object detection method. For depth prediction, we constructed and trained a U-Net [39] to map pixels input to depth in meters.

The overall process of extracting information from raw image is shown in 5.1. The model grounds raw visual input into both semantic and depth information which will be used as input for our egocentric planner. The semantic graph is constructed using the Networkx library [19]. The nodes of the graph are the location and orientation of the agent, and the edges represent the possible movements. We use bidirected graphs to distinguish movement in opposite directions. The conversion from learned graph to planning problem is done with Tarski with output specified in PDDL [36]. Our agent uses object detection to gather object and depth information from raw RGB images and stores this information in the spatial graph as a simple lookup. For simple object navigation tasks, our agent can directly navigate to any previously discovered object stored in the graph after a period of exploration. For more complex pick-and-place, we convert the planning problem to an egocentric alternative using methods specified in Chapter 4.2.

5.2.1 Object Detection

Our object detection method is based on YOLO [37]. It takes a raw image as input and outputs detected objects distribution with bounding boxes. Compared to two-step methods, such as RCNN [20], it is faster to train and easier to deploy. We adjusted the original network by adjusting the last two layers to fit the AI2-THOR object categories. We initialize the network with weights pretrained on the COCO dataset. We use the adam optimizer with a learning rate of 0.0001 during our training. The dataset used is generated with random configurations in AI2-THOR environments. To ensure generalizability, the training object configurations are not present in the graph construction phase.

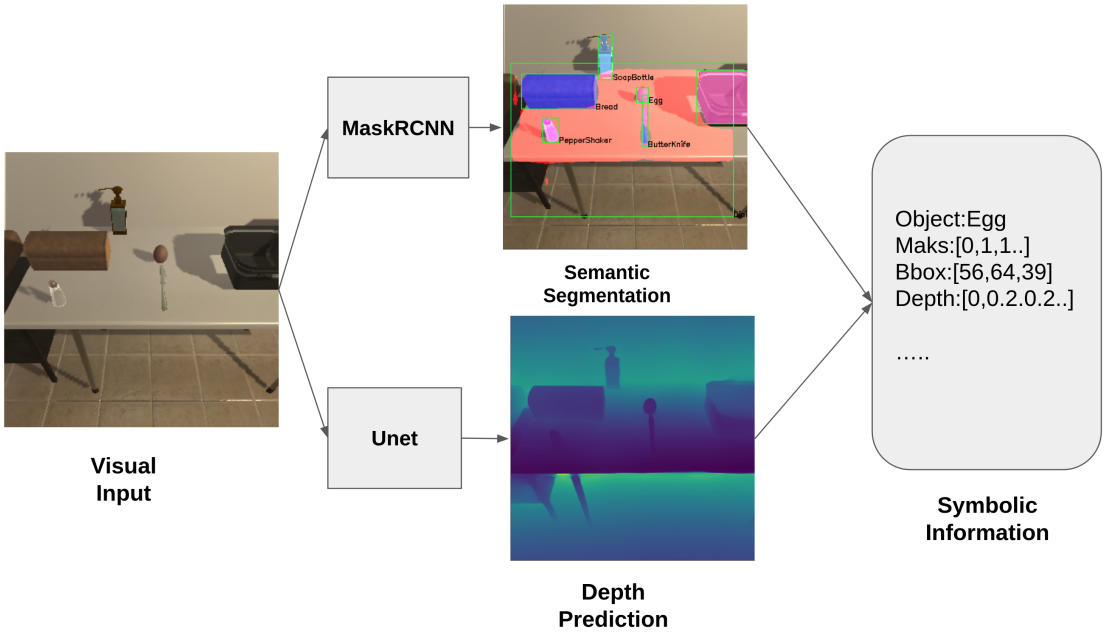

Figure 5.1: Information Extraction form Raw Image

5.2.2 Depth Estimation

We use a U-Net architecture to train a depth prediction network to estimate the distance between the agent and a target object. While U-Net has been used chiefly for image segmentation in medical data, it has also shown effectiveness for depth estimation [39]. We use a 10 layer architecture with a 5 layer convolutional encoder and an upsample decoder to mirror the encoder. We set the kernel size to 1024, 512, 256, 128, and 64 for both the encoder and decoder. The output is mapped to a 1D depth map range from 0 to 5 meters. Each layer uses ReLU activation with no additional dropouts. The network is trained with a batch size of 8 images. We use the Adam optimizer with a learning rate of 0.001.

5.2.3 Spatial Graph Generation

The spatial graph serves as the memory of the agent during exploration. We encode location as the key for each node and visual observations as values. The edges are actions taken by an agent. The four actions we included are MoveAhead, MoveBack, TurnLeft and TurnRight. The movement actions have a step size of 0.25 while the turning actions have a turn angle of 90°. For object navigation tasks, we gave the agent a budget of 1000 steps to explore. We use a simple navigation strategy where if the agent can observe an object of interest, it will move closer to that object. If no objects of interest can be observed, the agent will navigate to an unknown location following a uniform random distribution. The probability of whether an object is present is calculated using YOLO during each step. To make exploration more efficient, we implemented a queue to store locations visited. We also adopted a breadth search algorithm to facilitate exploration. The learned graph can be searched using any graph based algorithm for finding the optimal path. We used Dijkstra's algorithm for path planning as the default method [13]. We traversed through five environments to generate spatial graphs. An example of a generated graph is shown in Figure 5.2. On average, each graph has around 1231 nodes. However, only around 20 nodes have reachable objects. We setup a random walk agent to compare with our neural-symbolic agent. This random walk agent has equal probability of choosing each of the four actions. For pick-and-place task, the graph is generated on the fly without exploration. The agent convert the graph into a planning problem during each step and replan after each action is taken. This is done to simulate more realistic use-case where exploration needs to be done in concurrent with an agent's planned actions.

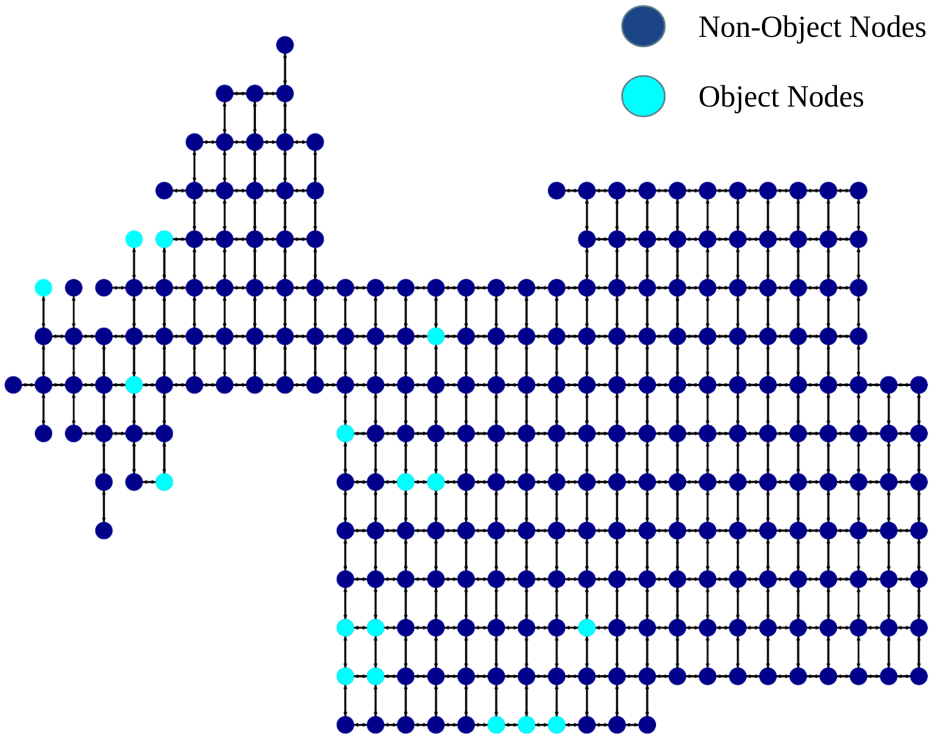

Figure 5.2: Example of a Learned Spatial Memory Graph

5.2.4 Egocentric Planning

The spatial graph is converted into a egocentric planning problem specified in PDDL. We then leverage the techniques in Chapter 4 to convert the spatial graph into a planning problem solvable via replanning. The outputs of the egocentric planner serve as policies to guide our agents. The goal is manually specified by the user in a PDDL syntax. When the Egocentric Planner determines that information stored in the spatial graph is insufficient to achieve the original goal, the planner automatically produces a policy for exploration and information gathering.

5.3 Experiments

We use the AI2-THOR environment [29] to setup our experiments. Both simple object navigation and pick-and-place tasks are conducted to assess our model performance. The environments we used to conduct our experiments are unseen by the agent because they are not used in the training set for semantic segmentation and depth predictions. For object navigation tasks, our experiments include three object categories in five virtual kitchen environment. We evaluated our approach using two criteria: the success rate and the average path length. We define success rate as the percentage of successful runs where our agent is able to find the target. The average path length is defined as the number of actions our agent needs to find a particular object. For the baseline, we compared our agent against a random walk agent with uniform distribution among available actions. The experiments are repeated five times, and the average is calculated. The pick-and-place tasks are designed to evaluate our model's performance on sequential decision tasks. In addition to finding the object, the agent needs to perform a "pick" action on a target object, navigate and locate a receptacle object and perform a "put" action while holding the target object. We use a subset of the ALFRED challenge [42] which include 3 "pick-and-place" tasks, each in 5 different virtual home environments. Unlike the object navigation tasks, no exploration budget is given for these tasks, and the agent will need to explore while planning for the next sequences of actions. An example of the ALFRED dataset is shown in Figure 5.3if In addition to the success of each task, we also evaluate our object recognition and depth prediction neural networks to better understand our performance bottleneck. The segmentation network uses the percentage of correct labels produced by the network, and the depth prediction is evaluated on whether an

object can be reached based on the average distance of an object.

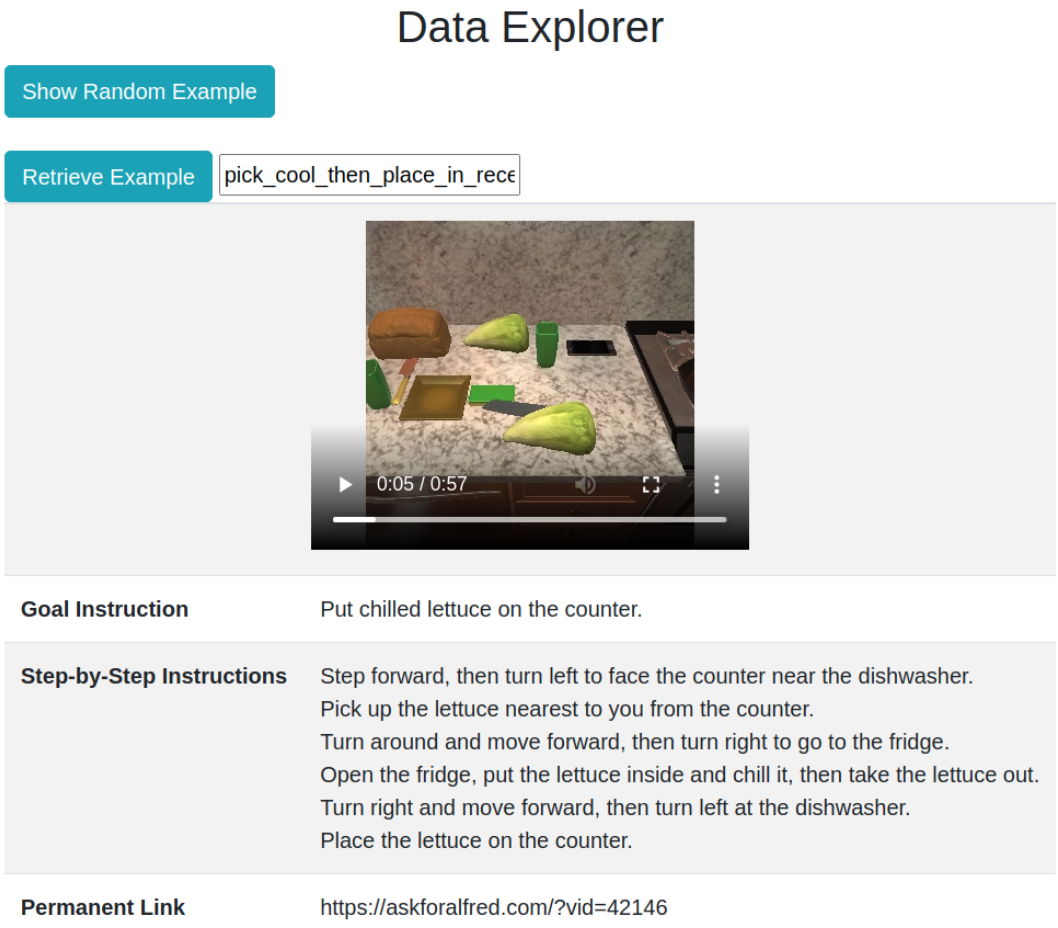

Figure 5.3: Example of ALFRED dataset[42]

5.4 Results

5.4.1 Semantic Segmentation and Depth Prediction

We use an 80/10/10 split to train both our object recognition network and our depth prediction network. Our training set contains around 400,000 images with both object

and depths labels. Both test and validation set contains around 50,000 images each. All images and labels are generated by the AI2-THOR simulator. The validation and test set are all unseen environments for the agent. We split validation and test based on different rooms used. We calculate the percentage of correct labeled objects as evaluation criteria to measure the object recognition network. An object is considered correctly labeled if the bounding box predicted has more than 80% overlap with the ground truth with the correct object category prediction. We trained our model for 20 epochs and achieved 92% accuracy on the test set and 89% accuracy on the validation set. Results can be seen in Figure 5.4. For our depth prediction network, we define accuracy as a binary classification task to determine whether an object is reachable or not. We use the average pixel-wise distance within a bounding box to estimate the distance between the agent and object. When the distance is smaller than 1.5 meters, we classify an object as reachable and unreachable otherwise. The network was trained for 50 epochs and achieved an 82% accuracy on the validation set and 86% on the test set. Results can be seen in Figure 5.5. Although we observed that validation accuracy continued to decrease after training, we decided to stop due to computational and time constraints.

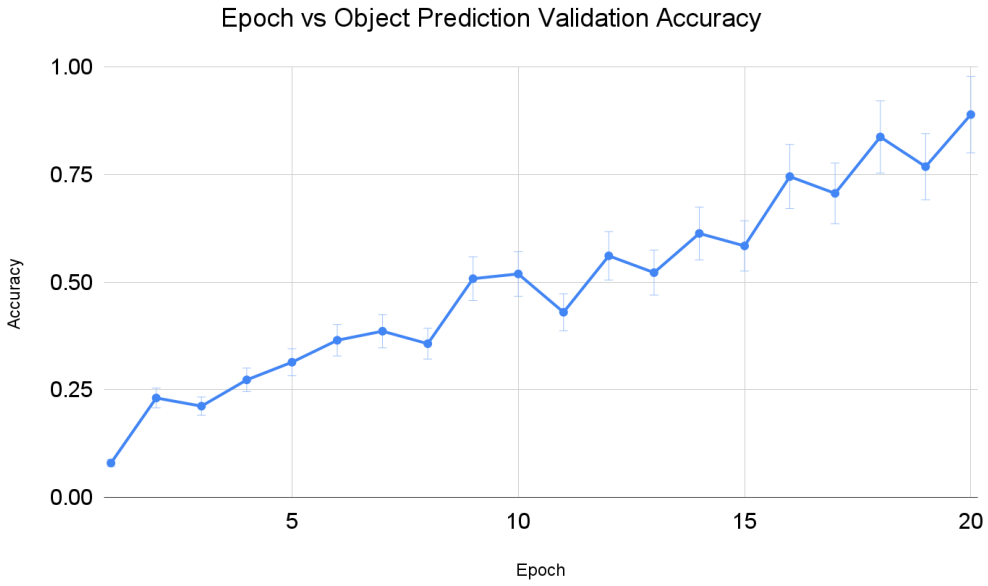

Figure 5.4: Object Recognition

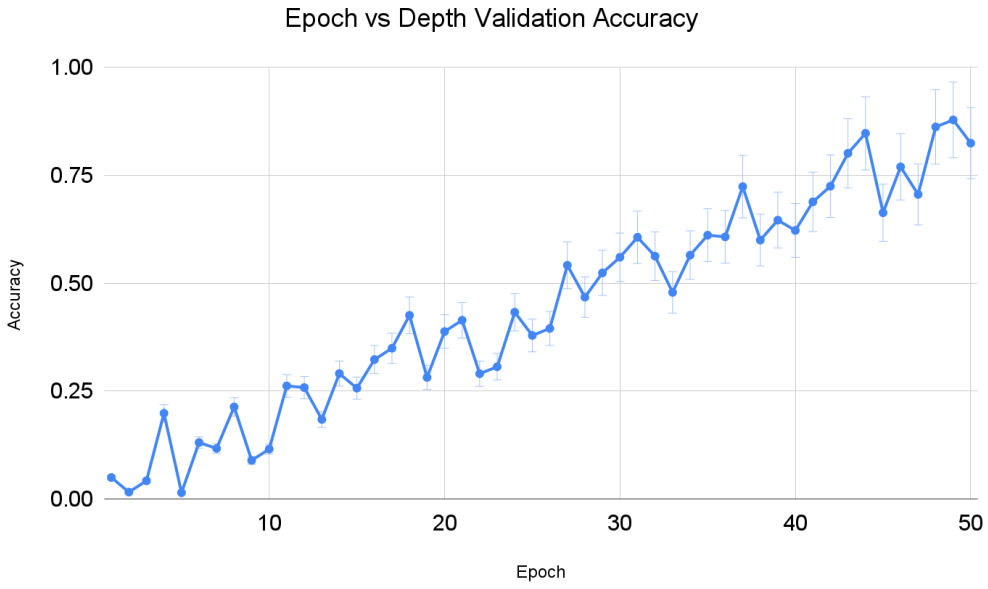

Figure 5.5: Depth Prediction

Object	Room 1	Room 2	Room 3	Room 4	Room 5
Apple_Acc	100%	100%	100%	100%	100%
Lettuce_Acc	100%	100%	100%	100%	100%
Tomato_Acc	100%	100%	100%	100%	100%
Apple_Len	21	23	20	27	25
Lettuce_Len	21	26	23	21	21
Tomato_Len	25	29	22	21	25
RandWalk_Acc	0%	66%	0%	0%	66%
RandWalk_len	6710	876	4429	951	2328

Table 5.1: Navigation Results

5.4.2 Object Navigation

Table 5.1 outlines the results of each object navigation tasks in each room, and Table 5.2 summarizes the results of our model's overall performance. Our agent is able to discover all target objects in all of the settings tested. The test results on the path length also confirm the effectiveness of our neural-symbolic method. The average length is 23 actions, which is slightly more than the optimal lengths of 19. Our method is reaching the optimal theoretical limit, which would require significantly more data and training steps in end-to-end RL approaches. When comparing our method to Random Walk, we can see a drastic improvement. An agent randomly searching in an environment with a given movement budget of 2000 can find an object 26.4% of the time. The average length for Random Walk agent is 3069, which takes two orders of magnitude times more actions on average than our agent. Our results demonstrate a great deal of promise in using neural-symbolic approaches for the embodied agent setting.

	Average Length	Success Rate
NS Agent	23	100%
Random Walk	3069	26.4%
Optimal	19	100%

Table 5.2: Summary of Object Navigation Results

5.4.3 Pick-and-Place

For pick-and-place tasks, we evaluate our agent's performance based on the completion of both partial and complete goals with their step counts. A complete goal is achieved when an agent is able to successfully pick up an object and move it to the corresponding receptacle. A partial goal is completed when an agent discovers either the correct object or the correct receptacle. To compare the length of each episode, we count the number of steps it took our agent to complete each task with a maximum budget of 1000. We also record the number of action failures and their corresponding failure categories. The results for each room are shown in Table 5.3 and failure breakdown is shown in Table 5.4. On average, our agent takes 128 steps for each episode with 16 action failures. An action fails most frequently when our agent moves into an obstacle, accounting for 71% of the total failures. Other failures occur when our depth or object detection network produces an incorrect prediction. The failures suggest that our model performances are bottle-necked by our neural network performances. We believe better classifiers and more data would help improve our results. Overall, we successfully achieved 86% of our partial goals and 60% of our complete goals. Random walk agent is unable to achieve any of the experiments under 10000 steps. The baseline provided by the ALFRED dataset all has a success rate under 5%, and pure end-to-end methods on the public ALFRED leaderobard

Object	Room 1	Room 2	Room 3	Room 4	Room 5
Avg Steps	78	122	52	255	134
Parital Goal	66%	100%	100%	66%	100%
Comp Goal	66%	33%	100%	33%	66%
Avg Failtures	12	21	7	24	17
RandWalk Acc	0%	0%	0%	0%	0%
RandWalk len	10000	10000	10000	10000	10000

Table 5.3: Pick-and-Place Results

are all below 10%. These methods are often very sophisticated and require weeks of training and hyper-parameter tunning on large GPU clusters. Our models are trained only using an Nvidia RTX 2070 GPU and an Intel i7 CPU. Because we disentangle object segmentation and depth prediction from the overall policy learning, the methods outlined can be used for zero-shot generalization in new tasks. Although the experiments are designed to evaluate our neural-symbolic approach, there are still significant overlaps between our experiments and tasks provided in ALFRED. Our overall goal completion rate of 60% suggest that our hybrid approach significantly output perform SOTA end-to-end models which are all sub 10%.

Failure Category	
Hit Obstacle	71%
Fail to Pickup	5%
Fail to Place	7%
Depth Error	17%

Table 5.4: Failure Breakdown

5.5 Conclusion

We outlined a neural-symbolic method for discretized object navigation tasks in the AI2-THOR environment. Our results demonstrate that a hybrid model can perform near-perfect object navigation tasks in simple home environments. In addition to object navigation, we also evaluated our results on long horizon sequential decision tasks. Our experimental results show significant performance advantages over end-to-end baselines. Overall, we demonstrated the effectiveness of a hybrid approach and have shown the feasibility of integrating neural networks with a symbolic reasoning engine. This work serves as a first step to building more advanced neural symbolic agents.

Chapter 6

Disscussion

To build an embodied agent that can act autonomously in unknown environments is not a trivial task. This thesis builds on many existing work in both deep learning and automated planning approaches. The two sub-fields we focus on are embodied agent navigation and planning approaches over partially observable environments. This section introduces previous works of agent navigation and then survey the most prominent partially observable planning methods. We outline and contrast existing work with our neural symbolic approach.

6.1 Embodied Agent Navigation

Agent navigation can be broadly categorized into geometric and learning based agents. Geometric based agents often require sensing and constructing a map of the environment using depth sensors. Once a map is constructed, localization is conducted in reference to the global map. On the other hand, learning-based agents use neural networks trained on expert trajectories to produce navigation policies. Inputs for these agents are often mixtures of visual and depth information. Rather than having an explicit map, learning based agents embeds geometric information in the weights of

neural networks. Our method can be considered a mixture of geometric and learning based systems. We use a graph to represent geometric between affordances which are extracted from raw visual input using neural networks. Our works address both learning efficiency and adaptability in existing systems.

6.1.1 Geometric based Navigation System

Traditionally, embodied agents require a map of the environment to conduct navigation. The map serves both as a way to localize an agent and a way to generate a path to the goal. Simultaneous Localization and Mapping, or SLAM, refers to a set of techniques to create a map of the environment using sensors on an agent. SLAM is the most common methods for traditional agent navigation tasks. It different from global navigation systems, such as GPS, because a SLAM agent generates the environment map with respect to itself [51]. SLAM is used for both map generation and agent localization. When an agent is navigating through an environment, the information is gathered through sensory. This information will be converted into a map that is used for path planning. For example, Thurn et al. used a grid generation technique to generate indoor rooms and hallways [50]. Gross et al. [17] used a laser-guided depth sensor to align the map with objects. Sometimes these generated maps are not grid-based. More recent SLAM algorithms focus on generating 3D maps rather than 2D. These maps are more suitable for realistic situations where elevation is often a big part of navigation. For example, Kummer et al. [30] uses 3D representations generated from image and depth sensors to map parking lots. For semantic navigation tasks, object information are often projected onto a global map, which could result in losing information required for egocentric actions. In contrast, our methods uses

a graph representation of affordances and store egocentric information in the nodes. This makes our methods more suitable for tasks beyond navigation.

6.1.2 Learning based Navigation Systems

Learning semantic navigation is a relatively new topic compared to geometric approaches. Deep learning has taken over the area of semantic navigation with the recent advancement in neural networks. Most deep learning-based approaches are being used for learning navigation policies via raw RGB images. Point-goal navigation is a form of semantic navigation that uses point location rather than an object. Wijmans et al. have used the end-to-end RL method to achieve state-of-the-art in point-goal navigation [53]. This work proposed a distributed RL system that can scale up to billions of training examples. However, these methods are very computationally expensive and require billions of frames to achieve state-of-art performance. Parisotto et al. has demonstrated that having a structured memory in RL agents helps with data efficiency [35]. They propose a location tensor as the memory structure that is updated using reader and writer processes found in the Neural Turning Machine. Using similar ideas, Tamar et al. proposed the Value Iteration Network to solve path planning problems on a local level [48]. Bhatti et al. [4] used a learned SLAM map to play Doom with RL agents. However, a fixed-size tensor is not the only way to store memory. Wu et al. used a generative model, inspired by Generative Adversarial Network (GAN), to learn a latent representation of the environment as a graph neural network [54]. It takes inspiration from human's ability to localize and imagine a virtual environment based on memory. Similarly, Yang et al. uses a pre-trained graph neural network to learn object relationships among objects[55]. These works

demonstrated that the learned object prior could help agents learn a policy without relying on recent observations. However, all these end-to-end methods suffer from the same problem of high data dependency and poor generalizability which our approach does not suffer from. Also these end-to-end systems are not adaptable for simple environmental shifts where are agents can conduct zero-shot learning in rearrange environments.

6.2 Planning under Partial Observability

Defining the environment in an egocentric manner is important to many agent based applications. Charniak demonstrates the utility of the egocentric view in reinforcement learning in a Grid World setting (2020). They show that the egocentric view improves the learning algorithm's ability to apply the learned policy to new problems never before seen during training. Zhang et al. proposed an egocentric vision-based assistive co-robot system (2013). This work allows humans to actively engage in control loops via egocentric camera and gesture input. Bertasius et al. presented a generative adversarial network model that use first-person images to generate a realistic basketball sequence via egocentric motion planning (2018). All these defined planning and egocentricity in their respective domain-specific settings. In contrast, our method focuses on egocentricity in the classic domain-independent planning setting.

There exists work on planning under partial observability, such as conformant, contingent, and epistemic planning. However, we distinguish egocentric planning from these partial observable planning settings with regards to the use of the belief space. Conformant can be interpreted as classical planning in belief space, and contingent planning adds uncertainty to the observable state space and can be formulated as

and-or search problems in the belief space [7, 22, 23]. Epistemic planning represents belief states as epistemic states which can be reasoned using epistemic logic [5]. Planners constructed for all of these partially observable planning settings often require explicitly defining the possible initial belief space. This requires the users of these planners to have knowledge of all possible objects that will be present in the planning problem, *in advance*. However, in many planning problems, an acting agent might not have access to all unique objects in advance, and new objects and relationships need to be discovered during exploration. For example, an egocentric agent that is navigating in a 2D grid might not know all possible "location" objects to formulate a proper belief state. The approach we take here is to iteratively convert information the agent has observed as a fully observable planning problem to determine whether the original goal can be reached or more exploration is needed. However, the fully observable planner we used can be replaced by conformant, contingent, or epistemic planners when dealing with uncertainties with agents, actions, or the environment. That is to say, those areas of planning under partial observability are *complementary* to our work, and may be incorporated in future work.

Some previous works have studied planning in open worlds, a similar class of problems to the egocentric problems we define in this paper. Talamadupula et al. used a hindsight optimization method to solve planning problems in partially observable worlds (2010). The proposed method uses a prior distribution to generate and aggregate samples of close-world problems that can be solved using an off-the-shelf planner. Kiesel et al. proposed a novel partial-satisfaction goal construct that allows predefined objects to be discovered via replanning (2012). James et al. presented a framework that derives egocentric views of planning problems for the purpose of

learning portable representations, sufficient for planning, of classes of tasks (2019). These representations are learned from traces of actions and transitions. Our work, however, derives these views from the planning problem's pansophical description. The advantage is our approach does not require training data. Jiang et al. introduces a method to identify and reason over objects in an environment via a database (2019). The approach converts open world problems to close world settings via hypothetical instances of unknown objects. All these works adopted assumptions about open-world problems that we expose via our approach to deriving egocentric problems. However, these problems require complete reformulation of close world planning domains via their respective specifications. To the best of our knowledge, we are the first to propose a method to semi-automatically convert classical planning problems into an egocentric alternative with relative ease.

Chapter 7

Conclusion

7.1 Summary

The overall objective of this research is to develop a neural symbolic framework for embodied agents tasks. We chose to use automated planning models specified in PDDL as high-level reasoning systems with environmental information learned through semantic segmentation neural networks. The proposed framework uses an object-centric perspective that assumes the world is made of objects and their relationships. The focus on objects enables us to build semantic location graphs, which bridge neural perception and symbolic planning modules. Our work also proposes a new open-loop replanning approach that focuses on the egocentricity of planning agents by interleaving exploration and planning. We tested our method in semantic navigation tasks in virtual environments specified in AI2Thor. Our results demonstrate the proposed framework is sample efficient with explainability similar to traditional symbolic methods.

Chapter 4 outlines our method of converting classical planning problems to egocentric alternatives. We use an object-centric representation of the state space by

formulating the world as objects and their relationship. Our algorithm specifies a set of exploration goals and exploration actions to conduct information gathering when the original objective is deemed unachievable. Our method can solve several open-world problems where the existence and count of objects are unknown prior to planning. We evaluated our method on a variety of classical planning problems, and our results demonstrate that the proposed method is sufficient to address the egocentricity of classical planning problems in a semi-automatic manner.

In Chapter 5 we proposed a Semantic Spatial Graph to bridge the representation gap between outputs of neural segmentation networks with the egocentric planner we presented in Chapter 4. We then tested a implementation of our planning-based neural symbolic framework on the object navigation tasks in a virtual home environment built with AI2-THOR. We chose YOLO as the sensory module to extract object information from an agent's egocentric visual input. The information is stored in an expanding Semantic Spatial Graph, which stores both location and object information in its nodes. The graph is then converted to a planning problem as initial inputs of our proposed Egocentric Planner. The planner then monitors an agent's observation and knowledge of the state space to goal-oriented explorations. Through empirical experiments, we have shown that our proposed framework can achieve near-optimal policies for simple object navigation tasks. We also tested our methods on long-term sequential decision tasks and achieved results significantly better than existing end-to-end approaches on the ALFRED dataset.

Our framework serves as a crucial first step towards planning-based embodied agents that can act autonomously in unknown environments. We have successfully demonstrated the feasibility of a hybrid neural-symbolic system. Furthermore, we

addressed problems of egocentricity in automated planning methods and proposed an object-centric way to bridge the representation gap between neural and symbolic methods. We hope our contribution in this thesis encourages further development of hybrid neural symbolic systems.

7.2 Limitations

Due to the recent successes in deep reinforcement learning, embodied agent tasks are often framed using an RL framework where rewards are hand-engineered for each task. However, most state-of-the-art RL methods require billions of sampled steps to learn a feasible policy for embodied problems such as object navigation. The hardware requirement to reproduce these deep RL methods in our setting is beyond the capability of our equipment. Thus, we chose to compare our results against what is being reported rather than reproducing these RL models. Another limitation is the complexity of the classic planning benchmark we used in the egocentric planning framework. These classic benchmarks are simplified versions of real-life problems which are much more complex and task-dependent. These application-specific planning problems are inaccessible to us, and further adjustment of our approach might be required depending on the application. Finally, the testing environment we chose to use is AI2-THOR which uses discrete actions in virtual home environments. Thus our approach has not yet been tested on systems with continuous control. Although our framework is viable for higher-level reasoning tasks, integration with existing control systems is vital to deploy our method in real-life embodied applications.

7.3 Future Work

Although we have shown the feasibility of building a planning-based neural symbolic embodied agent for object navigation tasks, many open questions and areas of improvement need to be addressed for real-life applications. One of the future directions we would like to explore with egocentric planner is applying dead-end detection techniques such as [31] to avoid actions that cause dead-ends. We want to integrate the agent's original goal with exploration for more goal-oriented exploration strategies. In this work, we only tested our method on classical planning domains. These domains are not representative of real-life planning settings. We want to eventually apply our techniques in embodied egocentric agent design to solve planning problems such as in [42]. Another potential improvement is for the exploration strategy we used. Currently, we use the exploration action produced directly by the planner, which is often the closest unexplored node to our agent's current location. However, goal-oriented knowledge such as object occurrence could produce a much better exploration strategy than the naive approach. We want to explore learning this policy using heuristic search techniques or through RL-based methods in the future. Finally, we have only implemented our full method for object navigation tasks. One future direction is to solve more complex tasks that require higher-level reasoning. Tasks such as rearranging items in a room or performing daily chores such as cooking are currently unsolvable problems for end-to-end neural networks. We hypothesize that by converting a visual environment to a symbolic representation, we can perform the higher-level cognitive tasks using a logical reasoning system such as automated planning.

Bibliography

[1] M. Asai, H. Kajino, A. Fukunaga, and C. Muise. Classical planning in deep latent space. *CoRR*, abs/2107.00110, 2021.

[2] R. Barber, J. Crespo, C. Gómez, A. C. Hernámdez, and M. Galli. Mobile robot navigation in indoor environments: Geometric, topological, and semantic navigation. In *Applications of Mobile Robots*. IntechOpen, Rijeka, 2019.

[3] G. Bertasius, A. Chan, and J. Shi. Egocentric basketball motion planning from a single first-person image. In *CVPR*, pages 5889–5898. Computer Vision Foundation / IEEE Computer Society, 2018.

[4] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, and P. N. Siddharth. Playing doom with slam-augmented deep reinforcement learning. In *CoRP*, Dec 2016.

[5] T. Bolander. A gentle introduction to epistemic planning: The DEL approach. In S. Ghosh and R. Ramanujam, editors, *Proceedings of the Ninth Workshop on Methods for Modalities, M4M@ICLA 2017, Indian Institute of Technology, Kanpur, India, 8th to 10th January 2017*, volume 243 of *EPTCS*, pages 1–22, 2017.

[6] B. Bonet. Conformant plans and beyond: Principles and complexity. *Artif. Intell.*, 174(3-4):245–269, 2010.

[7] B. Bonet and H. Geffner. Planning with incomplete information as heuristic search in belief space. In S. A. Chien, S. Kambhampati, and C. A. Knoblock, editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems, Breckenridge, CO, USA, April 14-17, 2000*, pages 52–61. AAAI, 2000.

[8] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Auton. Agents Multi Agent Syst.*, 19(3):297–331, 2009.

[9] T. Chakraborti, A. Kulkarni, S. Sreedharan, D. E. Smith, and S. Kambhampati. Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In J. Benton, N. Lipovetzky, E. Onaindia, D. E. Smith, and S. Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 86–96. AAAI Press, 2019.

[10] E. Charniak. Extrapolation in gridworld markov-decision processes. *CoRR*, abs/2004.06784, 2020.

[11] J. Crespo, J. C. Castillo, O. M. Mozos, and R. Barber. Semantic information for robot navigation: A survey. *Applied Sciences*, 10(2):497, 2020.

[12] B. Dai, Y. Zhang, and D. Lin. Detecting visual relationships with deep relational networks. In *CVPR*, pages 3298–3308. IEEE Computer Society, 2017.

[13] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[14] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning. *CoRR*, 2020.

[15] M. Fox and D. Long. PDDL2.1: an extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.*, 20:61–124, 2003.

[16] J. Gibson. The theory of affordances. *Hilldale, USA*, 1(2):67–82, 1977.

[17] H. Gross, H. Boehme, C. Schröter, S. Müller, A. Koenig, E. Einhorn, C. Martin, M. Merten, and A. Bley. TOOMAS: interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In *IROS*, pages 2005–2012. IEEE, 2009.

[18] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba. Mastering atari with discrete world models. *CoRR*, abs/2010.02193, 2020.

[19] A. Hagberg, P. Swart, and D. S. Chult. Exploring network structure, dynamics, and function using networkx. *https://www.osti.gov/biblio/960616*, 2008.

[20] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988. IEEE Computer Society, 2017.

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR (Poster)*. OpenReview.net, 2017.

[22] J. Hoffmann and R. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS*, 2005.

[23] J. Hoffmann and R. I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artif. Intell.*, 170(6-7):507–541, 2006.

[24] S. D. James, B. Rosman, and G. D. Konidaris. Learning portable representations for high-level planning. *CoRR*, abs/1905.12006, 2019.

[25] Z. Ji, R. Qiu, A. Noyvirt, A. Soroka, M. S. Packianather, R. Setchi, D. Li, and S. Xu. Towards automated task planning for service robots using semantic knowledge representation. In *INDIN*, pages 1194–1201. IEEE, 2012.

[26] Y. Jiang, N. Walker, J. W. Hart, and P. Stone. Open-world reasoning for service robots. In J. Benton, N. Lipovetzky, E. Onaindia, D. E. Smith, and S. Srivastava, editors, *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*, pages 725–733. AAAI Press, 2019.

[27] Y. Katsumata, A. Taniguchi, Y. Hagiwara, and T. Taniguchi. Semantic mapping based on spatial concepts for grounding words related to places in daily environments. In *Frontiers in Robotics and AI*, May 2019.

[28] S. Kiesel, E. Burns, W. Ruml, J. Benton, and F. Kreinmendahl. Open World Planning via Hindsight Optimization University of New Hampshire. Technical report, University of New Hampshire, 2012.

[29] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. AI2-THOR: an interactive 3d environment for visual AI. *CoRR*, abs/1712.05474, 2017.

[30] R. Kümmerle, D. Hähnel, D. Dolgov, S. Thrun, and W. Burgard. Autonomous driving in a multi-level parking structure. In *ICRA*, pages 3395–3400. IEEE, 2009.

[31] N. Lipovetzky, C. Muise, and H. Geffner. Traps, invariants, and dead-ends. In *The 26th International Conference on Automated Planning and Scheduling*, 2016.

[32] D. V. McDermott. The 1998 AI planning systems competition. *AI Mag.*, 21(2): 35–55, 2000.

[33] D. Moratuwage, M. Adams, and F. Inostroza. δ-generalized labeled multi-bernoulli simultaneous localization and mapping with an optimal kernel-based particle filtering approach. *Sensors*, 19(10):2290, 2019.

[34] C. Muise. Planning.Domains. In *The 26th International Conference on Automated Planning and Scheduling - Demonstrations*, 2016.

[35] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *arXiv:1702.08360*, Feb 2017.

[36] M. Ramírez and G. Francès. Tarski, 2021. URL `https://github.com/aig-upf/tarski`.

[37] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[38] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, pages 779–788. IEEE Computer Society, 2016.

[39] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI (3)*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[40] M. K. Sarker, L. Zhou, A. Eberhart, and P. Hitzler. Neuro-symbolic artificial intelligence: Current trends. *CoRR*, abs/2105.05330, 2021.

[41] L. Serafini and A. S. d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. In *NeSy@HLAI*, volume 1768 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[42] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox. ALFRED: A benchmark for interpreting grounded instructions for everyday tasks. In *CVPR*, pages 10737–10746. Computer Vision Foundation / IEEE, 2020.

[43] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[44] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, 2017.

[45] T. Silver and R. Chitnis. Pddlgym: Gym environments from PDDL problems. *CoRR*, abs/2002.06432, 2020.

[46] J. Song, S. Kim, and S. Yoon. Alignart: Non-autoregressive neural machine translation by jointly learning to estimate alignment and translate. In *EMNLP (1)*, pages 1–14. Association for Computational Linguistics, 2021.

[47] K. Talamadupula, J. Benton, S. Kambhampati, P. W. Schermerhorn, and M. Scheutz. Planning for human-robot teaming in open worlds. *ACM Trans. Intell. Syst. Technol.*, 1(2):14:1–14:24, 2010.

[48] A. Tamar, Y. WU, G. Thomas, S. Levine, and P. Abbeel. Value iteration networks. In *NeurIPS*, Dec 2017.

[49] F. Teichteil-Königsbuch and P. Fabiani. A multi-thread decisional architecture for real-time planning under uncertainty. In *3rd ICAPS'07 Workshop on Planning and Plan Execution for Real-World Systems*, 2007.

[50] S. Thrun. Probabilistic algorithms and the interactive museum tour-guide robot minerva. In *The International Journal of Robotics Research*, 2000.

[51] F. Uhan. A discussion of simultaneous localization and mapping. In *Autonomous RobotsAI*, 2006.

[52] A. Verma, H. Qassim, and D. Feinzimer. Residual squeeze CNDS deep learning CNN model for very large scale places image recognition. In *UEMCON*, pages 463–469. IEEE, 2017.

[53] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra. Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *ICLR*, Sep 2020.

[54] Q. Wu, D. Manocha, and K. X. Jun Wang. Neonav: Improving the generalization of visual navigation via generating next expected observations. In *AAAi*, Jun 2020.

[55] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi. Visual semantic navigation using scene priors. In *ICLR*, May 2019.

[56] E. Yee, M. N. Jones, and K. McRae. Semantic memory. In *Psychology Publications*, Mar 2018.

[57] J. Zhang, L. Zhuang, Y. Wang, Y. Zhou, Y. Meng, and G. Hua. An egocentric vision based assistive co-robot. In *IEEE 13th International Conference on Rehabilitation Robotics, ICORR 2013, Seattle, WA, USA, June 24-26, 2013*, pages 1–7. IEEE, 2013.

[58] W. Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *SSCI*, pages 737–744. IEEE, 2020.

[59] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *ICRA*, 2017.