University of Antwerp
Faculty of Science
Department of Computer Science
IDLab Research group

Learning to Navigate through Abstraction and Adaptation

Matthias Hutsebaut-Buysse

Submitted in fulfillment of the requirements for the degree of
Doctor of Science: Computer Science

Promotor
Prof. dr. ir. Steven Latré

Supervisors
Prof. dr. ir. Kevin Mets
Prof. dr. Tom De Schepper

Antwerp, 2023

University of Antwerp
Faculty of Science
Department of Computer Science
IDLab Research group

Members of the jury

Chair
Prof. dr. Bart Goethals
Universiteit Antwerpen, Belgium

Promotor
Prof. dr. ir. Steven Latré
Universiteit Antwerpen, Belgium

Members
Dr. Abdellatif Bey-Temsamani
Flanders Make, Belgium

Prof. dr. ir. Kevin Mets
Universiteit Antwerpen, Belgium

Prof. dr. Ann Nowé
Vrije Universiteit Brussel, Belgium

Prof. dr. Juan Hernandez Vega
Cardiff University, United Kingdom

Acknowledgments

When I first graduated from university and needed to figure out what to pursue next I was somewhat aware of the area of machine learning. However, the examples I encountered back then were about clustering Iris flowers and predicting house prices based on the number of bedrooms. So instead of pursuing a career in research at that point, I decided to give software development a try. During these years I quickly learned that developing most qualitative customer-facing software is mainly not about technology, but is really about understanding people and processes. I also learned that no matter how many automated tests you write, or hours of manual testing you do, users will always (mostly unwillingly) manage to find edge cases in order to break the rules you have defined.

After doing software development for a few years, one day I stumbled upon a blog post about a research area which was called Reinforcement Learning (RL). I was immediately blown away by the idea of instead of coming up with a set of rules, having the system figure out the rules by itself. In pursuit of learning more about RL, I spent many evenings trying to implement basic RL algorithms. I still remember the magical moments when my models managed to somewhat learn how to play a video game. I was convinced that this approach would be the future, and I decided to jump ship.

Through a friend, I heard that Prof. dr. ir. Steven Latré together with Prof. dr. ir. Kevin Mets were starting up a new RL lab, and they were looking for new Ph.D. students. I grabbed the opportunity and they became my supervisors. Being a research supervisor, however, is no easy job. You have to push aspiring future researchers in order to always go further, and obtain the highest quality and ethical standards while keeping everyone motivated, even when confronted with many failed attempts. I would like to thank Steven, for while also leading a large research lab, making the time to guide me, and allowing me to explore novel ideas on my own. I would also like to thank Kevin. Our conversations and your many critical reviews were of paramount importance in this journey. You really demonstrated through leading by example what being a researcher is all about. In the last few years, I also enjoyed the luxury of having Prof. dr. Tom De Schepper as an additional supervisor. I would like to thank Tom for the many often out-of-the-box and practical insights that he brought to the table. Through Tom, I also now know that

there is such a thing as an ”Oxford Comma”.

In the past few years, I had the pleasure of being part of the IDLab research group. As there are currently more than 100 people affiliated with IDLab, it would be impossible to individually thank each of you for the collaborations, the inspiring lunch conversations, and the support.

I would also like to thank the people at Flanders Make. As a programmer, I have developed a common fast-feedback loop habit. You write some code, see where it fails, and then iterate. If you utilize a similar strategy when working with hardware, you just might see some smoke. Working with the people from Flanders Make allowed me to rely on their hardware expertise and allowed me to develop novel approaches on real-world platforms. Through our various conversations, I learned a lot about the differences between the often very simplified virtual agents, and systems that actually operate in real-world environments.

My sincere thanks also go to all members of the jury for making the time and providing feedback on this dissertation.

Finally, I'm eternally grateful to be blessed with such an amazing family, friends, wife Céline, and kids Louise and Manon. Watching an AI system develop from random behavior into complex strategies is an unbelievable magic experience. However, this is still nothing compared to witnessing the human counterpart.

Antwerp, September 2023
Matthias Hutsebaut-Buysse

Samenvatting

Artificiële Intelligentie (AI) is niet meer uit het dagelijkse leven weg te denken en heeft onlangs grote doorbraken gekend. Deze doorbraken zijn echter vooral gesitueerd in de domeinen van Computer Visie (CV) en taalverwerking. Deze doorbraken werden mogelijk gemaakt door de overvloedige aanwezigheid van grote internet datasets die vaak grote hoeveelheden zorgvuldig gelabelde voorbeelden bevatten.

Helaas zijn er ook heel wat domeinen waarbij deze aanpak niet succesvol kan ingezet worden. Navigatie-taken waarbij het systeem bijvoorbeeld naar een punt moet navigeren, of een object moet terugvinden in een onbekende omgeving zijn bijvoorbeeld minder geschikt voor deze aanpak. Deze navigatiesystemen moeten namelijk interageren met vaak complexe omgevingen, die moeilijk weer te geven zijn met eenvoudige invoer/uitvoer combinaties.

Reinforcement Learning (RL) is een alternatieve werkwijze waarbij een AI-systeem zijn eigen dataset zal verzamelen door directe interactie met de omgeving. Helaas kent RL momenteel nog een aantal beperkingen. Een belangrijke beperking is bijvoorbeeld de hoeveelheid interacties met de omgeving die nodig zijn. Momenteel zijn er in heel wat gevallen nog erg veel interacties nodig om een gewenst gedrag te kunnen leren.

Dit maakt het momenteel nog moeilijk om RL onmiddellijk in te zetten in de echte wereld. Momenteel wordt er vaak gebruik gemaakt van virtuele gesimuleerde omgevingen om systemen te trainen. Helaas is het bij deze aanpak niet vanzelfsprekend om een agent die getraind is door middel van een virtuele omgeving onmiddellijk in te zetten in de echte wereld.

In deze thesis introduceren we een aantal nieuwe methodes die zich er op richten om RL efficiënter te maken in termen van het aantal benodigde interacties met de omgeving. Dit doen we door te werken op meerdere abstractieniveaus, en door voorgaand geleerd gedrag aan te passen. Het werken op meerdere niveaus laat het systeem toe om zowel op langere termijn te plannen, als acties voor te stellen om het plan effectief te kunnen uitvoeren. Voor een navigatieprobleem zou er bijvoorbeeld één niveau verantwoordelijk kunnen zijn om een volgende kamer om te bezoeken voor te stellen, een ander niveau kan dan verzocht worden om acties voor

te stellen om deze kamer effectief te bereiken. Deze aanpak (Hierarchical Reinforcement Learning (HRL)) is in sterke tegenstelling tot hoe RL momenteel wordt toegepast. Momenteel wordt er meestal met een volledig schone lei begonnen, en wordt er slechts op één abstractieniveau gewerkt.

Het werken op meerdere abstractieniveaus is echter geen nieuw idee en een grote hoeveelheid voorgaand onderzoek werd al verricht om dit mogelijk te maken. Er werden in het verleden al heel wat verschillende technieken en methodes voorgesteld. De eerste contributie van deze thesis bestaat uit het systematisch beschrijven en evalueren van bestaande aanpakken. Dit hebben we gedaan om het bestaande werk te verdelen in distinctieve categorieën. Door dit te doen waren we ook in staat om een aantal openstaande problemen te identificeren.

Om de kracht van het gebruik van abstracties aan te tonen, en om een eerste stap te zetten in het oplossen van de geformuleerde uitdagingen werd er een nieuwe methode genaamd Structured Exploration through Instruction Enhancement (SETIE) geïntroduceerd. Deze methode gebruiken we om een object terug te vinden in een procedureel gegenereerde huishoudelijke omgeving. De nieuwe methode heeft genoeg aan een beschrijving van het object geformuleerd in taal, en een RGB egocentrische observatie van de omgeving. Waar typische RL methodes in deze configuratie niet in staat blijken om een succesvolle strategie te ontwikkelen, is de nieuw geïntroduceerde methode in staat om dit probleem gedeeltelijk op te lossen door te handelen en te verkennen op verschillende abstractieniveaus.

De tweede aanpak die nagestreefd werd in dit werk om de mogelijkheden van RL verder uit te breiden formuleert een antwoord op de vraag öp welke manier kunnen we bestaande voorkennis selecteren en aanpassen om nieuwe taken sneller te kunnen oplossen?”

Om dit probleem op te lossen werd een eerste methode voorgesteld die gebruik maakt van steekproeven. De resulterende methode is in staat om in te schatten hoe goed een bestaand gedrag kan aangepast worden naar een nieuwe taak door enkel en alleen rekening te houden met de beschrijving van voorgaande kennis en de nieuwe taak.

Het nadeel van deze methode is dat er echter een significante hoeveelheid rekenkracht nodig is om de steekproeven uit te voeren. Deze extra rekenkracht zal pas te rechtvaardigen zijn na een redelijk aantal adaptaties. Om deze extra kost te vermijden hebben we een tweede alternatieve methode voorgesteld die gebruik maakt van een apart getrainde taalvoorstellingsmethode. We tonen aan dat door een manier waarop taal numeriek kan worden voorgesteld, we ook in staat zijn om na te gaan welke voorkennis het meeste geschikt is om een nieuwe taak op te lossen.

Het selecteren van voorkennis is echter slechts een deel van het taak adaptatie probleem. Een tweede deel van het probleem bestaat uit het efficiënt gebruiken van voorkennis. Wanneer een systeem simpelweg het vorige gedrag overneemt,

zal het ook geen nieuwe uitkomsten vertonen. Er moet dus een afweging worden gemaakt tussen wanneer voorkennis gebruikt kan worden, en wanneer het systeem op zoek moet gaan naar nieuwe oplossingen.

Om dit mogelijk te maken werd het *Disagreement Options* framework voorgesteld. In dit framework worden de uitkomsten van meerdere gerelateerde voorgaande strategieën geraadpleegd. De beslissing om voorkennis te gebruiken of niet wordt gemaakt op basis van het feit of de uitkomsten van de geselecteerde voorkennis consistent zijn of niet. In navigatieproblemen zal deze aanpak het systeem vlot tot bij het doel brengen, en toelaten om de laatste stappen zelf te ontdekken.

In deze thesis werd er ook onderzoek verricht naar het gebruik van RL in verschillende toepassingen. Dit werd gedaan door te focussen op de verschillen tussen gesimuleerde omgevingen en observaties geproduceerd door echte sensoren. Om dit probleem ten gronde te kunnen bestuderen werd een nieuwe omgeving geïntroduceerd die bestaat uit een fotorealistische scan van de IDLab - imec Antwerpen kantoren. Door zowel toegang te hebben tot de virtuele als de echte omgeving zijn we in staat om verschillen en problemen te kunnen identificeren tussen deze twee omgevingen bij het gebruik van RL.

Om verder het verschil tussen de echte fysieke wereld en de virtuele wereld te verkleinen werden er in deze thesis twee nieuwe aanpakken voorgesteld. In de eerste methode maken we gebruik van een *digital twin*. De tweede methode bestaat uit een nieuwe manier om om te gaan met Light Detection And Ranging (LiDAR) sensoren en een hiervoor speciaal ontwikkelde nieuwe simulator. We tonen aan dat op basis van deze simulator we in staat zijn om een strategie te ontwikkelen in deze simulator, en dat deze onmiddellijk kan ingezet worden in de echte wereld. Bovendien tonen we aan hoe deze methode kan worden ingezet om een ongeziene omgeving te verkennen in functie van een externe taakmodule.

Samengevat kunnen we stellen dat het werk dat voorgesteld wordt in deze thesis de mogelijke toepassingen van RL uitbreidt door gebruik te maken van verschillende abstractielagen, en door het mogelijk te maken om bestaande strategieën aan te passen naar nieuwe taken. Om dit mogelijk te maken kunnen verschillende contributies worden geïdentificeerd. Als eerste werd er een studie verricht van hoe er momenteel gebruik gemaakt wordt (en de huidige beperkingen) van abstractieniveaus in het gebied van RL. Als tweede werd het SETIE framework voorgesteld als een praktische methode om abstracties te leren en te gebruiken. Als derde werden twee verschillende methodes geïntroduceerd om voorgaande kennis te selecteren en te gebruiken. En tot slot introduceerden we twee methodes die het mogelijk maken om een RL systeem te trainen in een gesimuleerde omgeving, om dit vervolgens ook in te zetten in de echte wereld.

Summary

Artificial Intelligence (AI) has seen tremendous successes in the past few years. These breakthroughs have, however, mainly been situated in the areas of Computer Vision (CV) and Natural Language Processing (NLP). Breakthroughs in these areas have been fueled by the abundance of large internet datasets containing huge amounts of nicely labeled examples.

Unfortunately, this dataset approach is not very well suited for other types of tasks such as navigation tasks (e.g., navigating towards a set of coordinates, or searching for an object in an unknown environment). Navigation systems need to interact with often noisy environments which are often hard to model in terms of clean input/output labels.

Reinforcement Learning (RL) offers an alternative learning paradigm, which allows an AI system to obtain its own dataset through direct interaction with the environment. Unfortunately, RL is still plagued with its own set of problems. One major limitation of RL is its sample inefficiency. Currently, an RL-based approach needs large amounts of interactions with its environment in order to learn a satisfying behavior. This makes it impractical to utilize RL in real-world environments, and most often requires RL practitioners to train agents in simulated versions of the environment. It is however in most cases not straightforward how to utilize agents trained in simulation in the real world.

In this thesis, we propose a number of novel approaches which are able to increase the sample efficiency of RL approaches. This is done through working on multiple levels of abstraction, and through adapting prior related behaviors. Working on multiple levels of abstraction allows the agent to both define a long-term plan and come up with actual implementations of this plan. In a navigation setting one level could for example propose the next room to visit, while another level might be tasked with actually reaching this room. This approach, which is called Hierarchical Reinforcement Learning (HRL), is in sharp contrast to how RL is typically approached. Typical RL approaches start from scratch for each new problem and only work on a single level of abstraction.

Utilizing multiple levels of abstraction in RL is not a novel idea, and a lot of research has been conducted on this topic. However, there has been a copious

amount of different approaches and ideas. The first contribution of this thesis consists of a thorough survey and comparative study of the available approaches. This was done in order to categorize these approaches into distinct frameworks. This survey also resulted in a list of open research challenges, which were formulated in order to further advance the state of the art.

To demonstrate the power of utilizing abstraction and work towards solving the formulated open challenges, we introduce a novel method dubbed Structured Exploration through Instruction Enhancement (SETIE). In this approach, an agent is tasked with finding an object in a procedurally generated simulated domestic apartment setting. The agent only receives a description of the object formulated in language and only requires access to egocentric RGB observations of the environment. While regular RL approaches fail to learn any successful strategy, the introduced method is able to advance this problem by acting and exploring on multiple levels of abstraction.

The second approach of extending the capabilities of RL studied in this thesis resolves around the question of how we can efficiently select and use fully developed prior policies when learning novel related tasks.

The first contributed method in this setting takes a sampling-based approach. The resulting method is able to predict transfer performance solely by looking at the (language) instructions attached to prior developed policies, and a new task.

However, sampling task adaptations requires a lot of computing resources, which could probably only be amortized after a lot of different adaptations have been performed. An alternative approach to sampling is introduced in the form of utilizing a pre-trained word embedding in order to select a prior policy, which can be adapted to solve a novel task. The main demonstrated hypothesis utilized here is that instructions that are close in a language space, will also have a policy close in policy space (and thus adapt well into each other).

Deciding which priors to select is however only part of the task-adaptation problem. The second part entails the question of how to efficiently utilize these priors. An agent cannot simply fully greedily exploit prior policies, as in this case, it will not learn anything new. Thus, a delicate balance between exploring new strategies and exploiting prior knowledge is required.

The proposed *Disagreement Options* framework is able to handle this trade-off. In this approach, multiple priors are consulted, and decisions are made based on whether they agree or disagree on the next action. In an object navigation setting utilizing this scheme allows the agent to utilize a prior policy to navigate close to the object and then explore for itself how to reach the new goal object.

To extend RL systems towards real-world applications we study the sim2real gap in navigation applications in the final part of this thesis. In order to study this

problem, a baseline task is introduced which utilizes a photorealistic scanned mesh of the IDLab - imec Antwerp office floors. This task is utilized to study various problems encountered when deploying real-world RL navigation policies.

In order to further minimize this sim2real gap, and allow the deployment of real-world RL systems, there were two novel approaches introduced. In the first approach, a digital twin is utilized in order to perform navigation tasks using a real-world Automated Guided Vehicle (AGV), utilizing an RL model trained in simulation. The second proposed method utilizes a novel developed Light Detection And Ranging (LiDAR)-based warehouse simulator. This simulator allows training policies utilizing RL, which can in turn be directly deployed on real-world hardware. Finally, it is demonstrated how this approach can be utilized to explore a previously unseen building, or how to utilize it in order to perform directed exploration (e.g., navigation in function of reducing the uncertainty of a second separately trained model).

To summarize, this dissertation further improves the applicability of RL by harnessing the power of abstraction and by allowing policies to adapt from prior knowledge. In order to achieve this, different contributions were made: first, a comprehensive survey was conducted on the usage of abstraction in RL. Second, the SETIE framework offers a practical approach to using and learning abstractions. Third, two approaches for selecting prior knowledge were introduced together with a novel method on how to efficiently utilize priors. Finally, we focus on how to apply RL in real-world settings through training in simulation.

Inhoudsopgave

1 Introduction

1.1 Context

Imagine you are tasked with building software that autonomously would operate a cargo ship across the Atlantic Ocean. A lot of the day-to-day operations can probably be automated using regular software programs. Pieces of code can be written to operate the engines, perform navigation on the ocean, keep the freight at the right temperature, turn on the lights at night, and probably even detect and extinguish fires.

However, if it would be impossible to physically reach the vessel when issues pop up, would you not bring any humans aboard? Probably not. Similar to how commercial airplanes are mostly relying on software to carry out most of their operations, human pilots are still kept in the loop in order to handle unexpected situations. When looking at (old) sci-fi books and movies, we would have expected that these ships and airplanes should not require any human operators at all by now. So one could ask why it currently is still unfeasible to also program autonomously operating pieces of code to handle calamities.

When looking at typical pieces of programming code utilized today, it is possible to observe some commonalities. The first common property of commonplace programming code is that it typically works on structured data. Think of tables filled

with mostly numbers stored in (relational) databases and spreadsheets. A common banking transaction will typically remove some money from one row of a database table, and add the same amount to another row.

The second property of common programs is that the rules which are applied are mostly not very complicated, and if broken down into smaller submodules, can be well-defined by human programmers through the definition of various functions filled with conditional statements. Computers nowadays have become extremely fast and precise in applying these rules to huge amounts of structured data.

Unfortunately, a lot of tasks do not fit this paradigm very well, and require the software to work on unstructured data such as images or natural language. This is especially true when interfacing with humans. Additionally, it might be impossible for a human programmer to come up with sets of rules in order to reliably solve the task. This might be due to the sheer amount of rules required, the often subtle thresholds, or because we might simply have no clue what the solution should look like. The typical example often mentioned in this context is the problem of how to write a program capable of detecting whether an image contains a cat or dog when only having access to a numeric representation of the individual pixels within the image.

So what alternative to writing code do we have in order to autonomously solve complex problems which require the processing of unstructured inputs and outputs? One might argue that solving such problems requires a form of intelligence. Intelligence has been defined in various ways (Legg and Hutter, 2007). However, properties such as creative problem-solving and the ability to adapt to various novel environments and task variations are commonly mentioned. While intelligence has often been linked with how humans or animals behave, the research area of Artificial Intelligence (AI) (McCarthy et al., 1955) has been tasked with finding approaches on how to obtain similar intelligent behavior within a computer system.

One of the directions pursued in this field consists of building a *Thinking Machine* (Turing, 1950). Instead of defining the rules on how to solve tasks, a thinking machine could learn these rules by itself given enough examples. Turing proposed to base the thinking machine of a *child brain*, as it is potentially simpler in structure, but capable of developing into an adult brain through embodied interaction with the world.

In the past a lot of different algorithms have been introduced following this paradigm on datasets containing structured data. Deep learning-based approaches (LeCun et al., 2015; Schmidhuber, 2015) have also demonstrated of being capable of implementing thinking machines through processing large amounts of high dimensional unstructured data, often curated from various internet sources (e.g., ImageNet (Deng et al., 2009), COCO (Lin et al., 2014) or VQA (Antol et al., 2015)).

Due to the origin of these datasets, the type of intelligence obtained through them is often termed *Internet AI*.

Figuur 1.1: **Internet AI Example: Visual Question Answering**
Examples taken from the VQA dataset. Through numerous labeled examples and deep learning approaches this dataset can be utilized to train a system capable of answering questions on the contents of an image. (Figure reproduced from Goyal et al. (2017).)

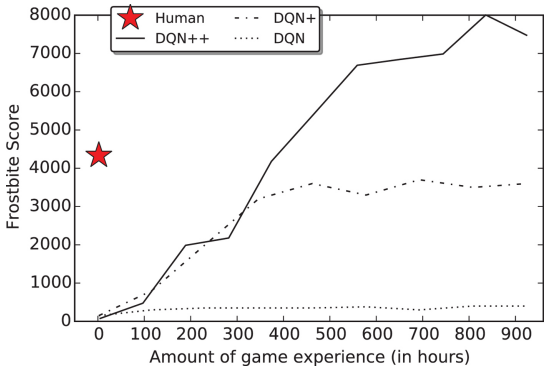
Current deep learning systems are capable of learning to solve different kinds of tasks by discovering complex patterns within these static collections of large amounts of labeled (unstructured) examples such as images, videos, or written texts. Even a generalist model (Reed et al., 2022) has been proposed, capable of handling different modalities and tasks (e.g., game playing, robotic manipulation, question answering) using the same trained model.

1.2 Problem Statement

1.2.1 Towards Human-like Learning

There is still more to intelligence that we would like to see in artificially intelligent systems than is currently possible. Humans still outperform AI systems in various complex tasks such as navigating in a 3D space (Mishkin et al., 2019) or finding objects in buildings (Ramrakhya et al., 2022). Additionally, in tasks in which AI systems already have outperformed humans, we however still often see that humans require far fewer data samples in order to reach a reasonable performance level. Figure 1.2 displays this phenomenon in the classic *Atari 2600* game *Frostbite*. Diuk et al. (2009) observed similarly that humans (especially those ac-

quainted with video games) utilized a significant amount of prior knowledge when comparing humans and AI agents in a *Taxi*-game.



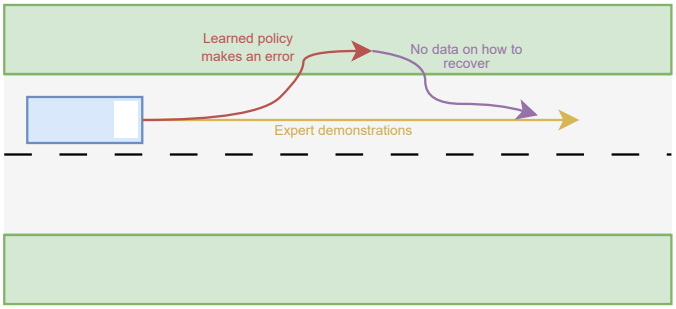
Figuur 1.2: **Human vs AI Training Performance on the Frostbite Video Game** Human performance compared to learned agents on the Atari 2600 video game Frostbite. While the learned agent is able to outperform the human, the human is almost instantly able to abstract the game mechanics and adapt prior skills. (Figure reproduced from Lake et al. (2017).)

Lake et al. (2017) argues that in order to reach truly human-like learning and thinking AI systems should (1) build causal models of the world that support explanation and understanding; (2) ground learning in intuitive theories of physics and psychology; and (3) harness compositionality and learning-to-learn to rapidly acquire and generalize knowledge to new tasks and situations.

It is however still mostly unclear how to exactly implement these requirements. As solving all these requirements will most definitely not fit inside a single doctoral thesis, the scope of this thesis will focus on answering the question of **how to harness compositionality** and **how to implement learning-to-learn mechanisms**. Compositionality resolves around the idea that new representations can be constructed through combining primitive elements. Think of individual Lego bricks which can be stacked in order to build the most diverse constructions. This concept of compositionality is naturally related to the concept of learning-to-learn, in which prior knowledge can be seen as the Lego bricks which facilitate the acquisition of new skills.

1.2.2 Efficiently Obtaining the Right Experiences

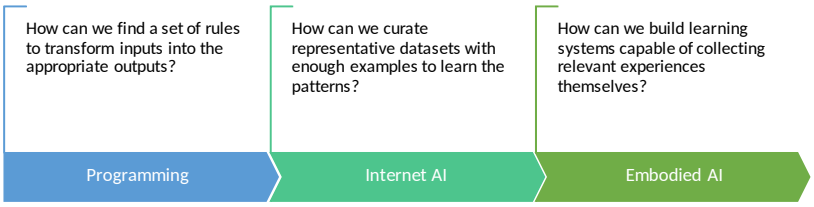
Deep learning has amounted to impressive results in the areas of computer vision and natural language understanding, partly through the availability of large datasets, containing labeled examples.



Figuur 1.3: **Mismatch between samples observed during training and inference**
If an autonomous vehicle would be trained on expert demonstrations, it might not be capable to recover from mistakes, as it has no data on how to perform these recovery procedures.

While such *Internet AI* datasets could also be constructed for other types of tasks such as robotic navigation (Mo et al., 2018) or manipulation tasks (Dasari et al., 2019), this approach has not been as effective as in computer vision and language understanding. One simple reason for this, is that there currently are no internet-scale datasets available on robotic behaviors. However, if such datasets would be commonly available there would still be the issue of a **mismatch between the distribution of samples observed during training, and observations during evaluation** (Figure 1.3) (Kumar et al., 2019; Fujimoto et al., 2019) and **overfitting** (Fu et al., 2019). In order to work efficiently with current machine learning approaches, the dataset would need to consist of independent and identically distributed random variables. However, in a sequential decision-making context, this is typically not the case as actions and states influence future states.

Additionally, there is the related question of representation learning. **How can we represent the current state of the environment in order to come up with an optimal answer?** In a setting in which a robot is tasked with navigation, the traditional approach of representing the environment is through an occupancy map. However, such an occupancy map might not be the best representation in all cases (Levine and Shah, 2023). For example, tall grass might show up as occupied space, and thus inaccessible, while in reality, a robot could perfectly traverse the grass. The opposite might be true for a muddy puddle which will not be picked up as special inaccessible terrain by typical laser-based sensors but might bring the robot to a full stop in practice. A recent study (Partsey et al., 2022) has even presented evidence that explicit mapping may not be necessary for navigation at all. Blind agents with only access to an ego-motion sensor have been shown capable of building and utilizing implicit map-like representations of their environment solely through environmental interaction (Wijmans et al., 2023).



Figuur 1.4: **Evolution of AI Approaches**
Different approaches of obtaining intelligent behavior together with their main challenges. This thesis addresses the problem of how to move beyond Internet AI towards Embodied AI.

So how do we need to represent the world in order to successfully navigate? If we look at human intelligence (Ormrod, 1999) again, it quickly becomes clear that processing large amounts of labeled examples is not the only way humans learn about the world. So what other methods of learning can we use as inspiration in order to be more successful in the envisioned tasks? Research in developmental psychology (Smith and Gasser, 2005) has argued that intelligence emerges in the interaction of an agent with the environment and as a result of its sensorimotor activity. This *Embodiment Hypothesis* has been the basis of a subfield of AI called Embodied AI (EAI) (Figure 1.4).

This subfield is closely intertwined with Reinforcement Learning (RL). RL (Sutton and Barto, 2018) is simultaneously a problem and a class of solution methods that attempt at solving sequential decision-making problems. A key property of RL consists of trial-and-error learning through interaction with the environment. The idea of trial-and-error learning is closely related to the embodiment hypothesis. RL can be seen as a practical framework for obtaining intelligent behavior through interaction with the environment. For example, if one would like an agent to be capable of riding a bicycle, we could utilize a deep learning-based approach, and provide the agent with images or videos of people riding bicycles. If we instead apply RL on the same task, we would allow the agent to control a bicycle, and figure out for itself how to optimally control it through trial and error.

RL has seen tremendous successes in the past few years. Some examples include; outperforming humans in (video) games (Mnih et al., 2015; Silver et al., 2016; Vinyals et al., 2019; OpenAI, 2019a), controlling nuclear fusion reactions (Degrave et al., 2022), steering stratospheric balloons (Bellemare et al., 2020), molecular design (Simm et al., 2020), chip placement (Mirhoseini et al., 2020) and dexterous robotic manipulation (OpenAI, 2019b).

However, RL is still plagued with a lot of open problems as well. On a fundamental level **coming up with a good exploration scheme** has been deemed challenging. When do we utilize the knowledge we already have obtained, and when should we try a new action?

A second fundamental issue consists of finding an answer on **how to design a reward function?** If we would like the agent to learn how to behave in a certain way, there is the need to specify what exactly is good behavior. In the RL framework this is done by specifying a reward function. Ideally, such a function is capable of providing stable and dense feedback on how desirable each step taken by the agent is, given the state of the environment. Designing such a function is called reward shaping (Ng et al., 1999), and is very prone to error. For example, a drone might have learned to crash itself if the specified penalty for crashing is not in balance with the penalty for utilizing energy while navigating to a destination. Ideally, one would not have to engineer a complicated reward function. For most tasks, it is simpler to specify the reward function in a sparse and binary fashion. This would only provide a positive value for completing the entire task. While specifying the reward function only sparsely alleviates the complexity of reward shaping, a new issue of **how we can learn from sparse reward signals** arises.

The areas in which RL has been applied successfully are often areas in which samples are easy to obtain because the task at hand is either virtual by nature (e.g., a video game), or the environment can be simulated with a high level of realism. However, in order to advance to more real-world RL applications, **sample complexity** is one of the main problems hindering progress (Dulac-Arnold et al., 2019; Zhu et al., 2020; Ibarz et al., 2021). Utilizing RL based solutions still often requires too many interactions with the environment in order to learn the desired behavior, rendering RL a highly impractical approach.

In summary, in this thesis we will propose novel methods which will work towards allowing AI systems to learn more human-like by allowing them to harness compositionality and implement learning-to-learn mechanisms. In order to solve complex problems we allow the AI system to collect its own experiences through embodied interaction with the environment. To make this possible we focus on the problems of exploration, reward function specification, and sample complexity.

1.3 Hypothesis

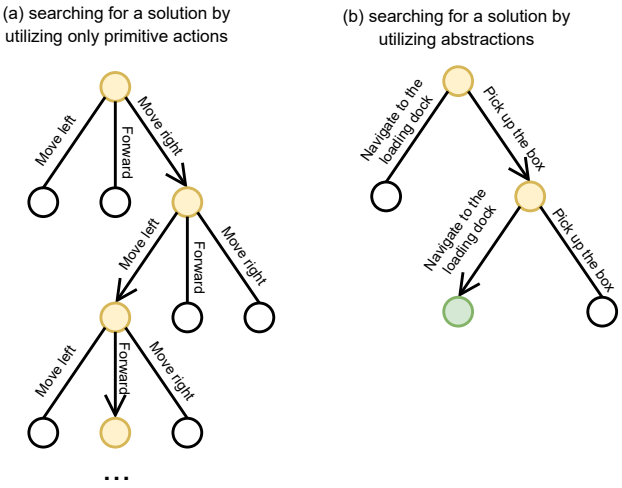
In the previous section, multiple core problems were identified for which finding solutions will be quintessential in order to further progress the current state of RL. To do this we formulate the following hypothesis:

> In order to extend the real-world potential of RL approaches, they need to become more sample efficient. This can be done by introducing compositionality through learning with abstractions, and by allowing the agent to efficiently adapt abstracted prior knowledge obtained in the past.

Training an RL agent could take days or even weeks (Silver et al., 2018; Vinyals et al., 2019; Wijmans et al., 2022b) depending on the complexity of the task. We believe that by focusing on working towards reducing the number of required interactions with the environment, and thus the overall training time, the potential applications will increase.

We believe that this will be possible by introducing both abstractions and allowing the agent to adapt to related (e.g., same environment, same sensors), but novel, problems (e.g., different task). The idea of working with abstractions, mapping a representation of a problem onto a new representation (Giunchiglia and Walsh, 1992), can be dated back to the very origins of AI (McCarthy et al., 1955). However, this approach is still in sharp contrast to how RL problems are typically approached. RL problems are typically approached by directly approaching tasks on the level of individual primitive actions and states. This however results in huge search spaces and makes credit assignment, figuring out which past individual actions actually made a difference on the final result challenging. This problem is further exacerbated if the reward signal is only sparsely defined. An example of this approach would be if you have access to a handheld controller in order to move a robot in small steps (e.g., move 15 cm forward, or turn 30 degrees to the left). You would repeatedly need to carry out a complicated set of action sequences. The only information you receive is whether you completed the task or not. Even finding a single example solution in this setting will require an unreasonable amount of trial and error. In order to figure out the pattern and learn a repeatable solution, you will probably require a lot of these examples.

The introduction of multiple layers of (learned) abstractions has been a key breakthrough in deep learning (LeCun et al., 2015). Utilizing many layers of abstractions on different levels has made many tasks possible within areas such as computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), and Natural Language Processing (NLP) (Sutskever et al., 2014). Instead of directly looking at values of pixels, it becomes easier to detect if a dog is present in a pic-



Figuur 1.5: **Searching for a solution through primitive actions vs abstracted actions** Searching for solutions (green node) in search trees with only primitive actions (arrows) quickly becomes intractable. Having access to the right temporally extended actions reduces the search space significantly.

ture if you have access to a (latent) high-level representation which might contain information about if the image contains dog ears, a tail, four paws, and fur.

We believe that similar utilization of learned abstraction within RL might significantly reduce the search space of potential solutions. In the example of the handheld controlled robot, the controller would now not contain individual primitive movement actions but would contain temporally extended actions which would group multiple primitive actions in order to make a signification impact on the environment. Buttons on the controller now might trigger behaviors such as *pick up the box* or *navigate to the loading dock*. In this setting, there is a much smaller set of possible solutions to explore (Figure 1.5).

The ability to reason on multiple levels of abstraction will be an important part of making RL more sample efficient. However, working with immutable abstractions is only a partial solution. It would be possible to learn new abstractions as new tasks require them. We however believe that it would be more sample efficient to start the development of novel abstractions by adapting existing related abstractions.

1.4 Research Questions

In order to validate the formulated hypothesis, we define the following five research questions:

1. **What are the various approaches in which abstractions are currently utilized in RL? What problems still need to be solved in order to allow efficient usage of abstractions?** The usage of abstractions is not a novel approach within RL. It is, however, a very broad field with various techniques introduced both before the advent of deep learning, and after deep learning became broadly adopted. In order to get a good understanding of what is currently addressed, and what are still open challenges, an extensive survey is required to further expand its applicability.

2. **Can a learned hierarchical approach extend the capabilities of navigation agents?** Non-hierarchical flat approaches often fail to make any progress, when the task requires long-horizon planning and memory. This is often the case in navigation tasks. In this type of task, an agent often needs to explore an unseen complex area in order to solve a given task. However, utilizing abstractions in this setting is also far from trivial. Questions such as how can different abstraction levels be rewarded, trained, explore, and communicate with each other, need to be carefully addressed. Additionally, if the task is specified semantically (e.g., find a specific object), the issue of grounding these instructions in the environment needs to be addressed. To solve this issue the agent will need to be capable of mapping instruction subjects to real-world concepts.

3. **How can an agent tasked with a new goal select prior knowledge in order to perform this new task more efficiently?** As plotted in Figure 1.2, a human player is able to achieve a certain level of performance on a new task (in this case playing the Frostbite game) with only a relatively small amount of practice. While the artificial agent in this setting is able to achieve an above-human level of performance, it requires far more hours of training in order to do so. These results can be explained due to the fact that the artificial RL agent started from a random initialization, while the human agent has a lifetime of acquired skills, which potentially can be adapted to the novel task. In order to emulate the usage of prior skills within an artificial agent, the question of, given a set of prior skills, which ones will be the most beneficial to this novel task should be addressed.

4. **How can the selected prior knowledge be utilized efficiently when solving novel tasks?** Selecting which prior skills to use is only the first step in an approach toward solving efficient task adaptation. A second question that should be addressed is the question of how this prior knowledge might

actually be utilized. A naive approach could consist of initializing a novel policy with the learned parameters from the selected prior knowledge. However, it was observed that within machine learning in general (Wang et al., 2019b) and RL (Taylor and Stone, 2009; Lazaric and Restelli, 2011; Rajendran et al., 2017) negative transfer performance can occur when pursuing this approach. More elaborate prior knowledge sampling approaches will be required in order to avoid negative transfer effects.

5. **How can an RL approach be utilized in the real world when taking into account current sample complexity limitations?** The results from addressing the previous problems and questions could have the potential of greatly improving the sample complexity of current RL approaches. If at some point RL approaches would require only a few interactions with the environment, they can potentially be directly trained on real-world observations. Until this might become possible, it will also be important to study alternative approaches in which sample complexity is not the performance bottleneck holding back real-world application of RL-based approaches.

1.5 Research Contributions

In the search for answers to the previously defined research questions, the following contributions were made:

1. **A survey on the integration of abstractions within the RL framework** (Chapter 3)
    - While prior work was mainly focussed on pre-deep learning-based approaches, this survey makes a bridge between classic approaches and novel deep RL based approaches.
    - The introduction of a taxonomy of the frameworks currently in use.
    - A comparative analysis of the different reviewed frameworks.
    - A comparative analysis of the main implementations within each reviewed framework.
    - In order to spark further research, within this research area, a list of open challenges was proposed together with hints of potential solutions.
2. **A hierarchical framework (SETIE) capable of utilizing abstractions in challenging object navigation settings** (Chapter 4)
    - The framework is capable of operating on two different levels of abstraction. The top-level learned planner is capable of reasoning and

exploring on a floor plan level, while the lower-level learned controller is capable of navigating by utilizing primitive actions. The agent only requires an egocentric RGB camera input.

- Instructions can be provided as simple language instructions. In order to facilitate language grounding a separately trained goal assessment module is proposed.
- A novel loosely coupled interface (instruction enhancement) is introduced in order to communicate between both levels using language.
- A partly procedurally generated environment was introduced. This environment was specifically developed in order to fundamentally study object navigation in domestic settings.

3. **An approach to select prior knowledge in order to perform task adaptation** (Chapter 5 and 6)

   - In order to select prior knowledge, an approach based on supervised learning and sampling of task adaptations was proposed, based on the instructions attached to various tasks.
   - A study was conducted on how well policies linked to different tasks were able to adapt to novel (related) tasks.
   - A related follow-up approach was proposed in which the supervised learning scheme was replaced by utilizing a pre-trained word embedding.

4. **An approach to efficiently sample from prior knowledge** (Chapter 7)

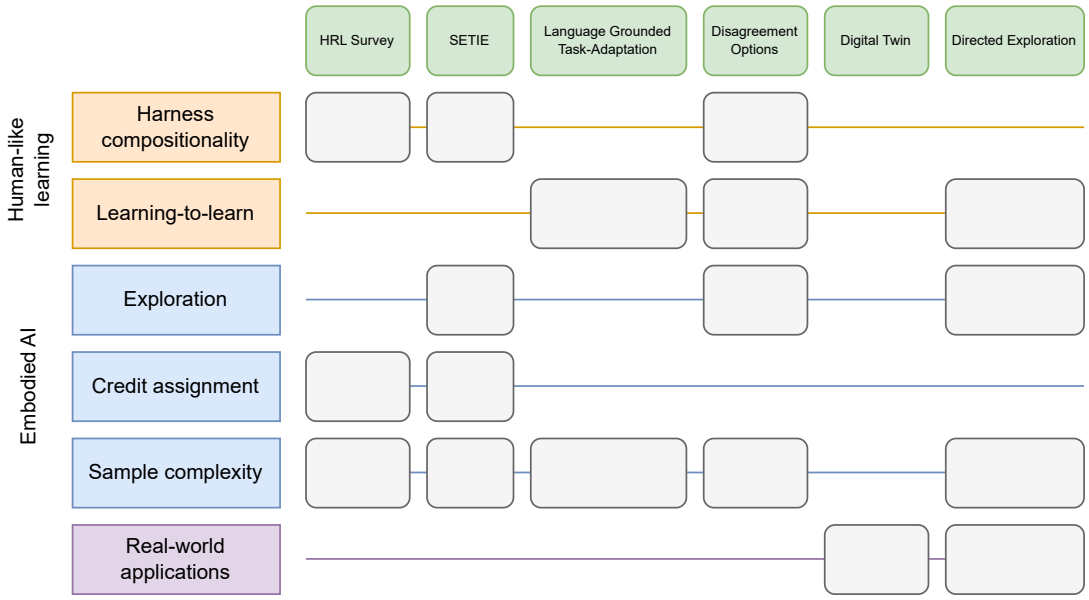
   - In order to answer the question of how to utilize the selected prior knowledge efficiently, a novel framework was developed capable of assessing when to utilize prior knowledge, and when to explore.
   - These exploit/explore decisions are made by a novel strategy based on the disagreement between the action distributions of the prior policies.

5. **The Beacon: A photo-realistic office simulator** (Section 8.2)

   - For studying whether utilizing a photorealistic simulator could allow a policy to be trained and then be utilized in the real world, a simulated virtual version of the IDLab - imec office floors was constructed.

6. **A digital twin based approach in order to utilize RL approaches in real-world environments** (Section 8.3)

   - In order to perform real-world point goal navigation through RL training an approach based on a digital twin was proposed. This digital twin offers a digital synchronized representation of relevant aspects of the real world, which can both be utilized during training and inference.



Figuur 1.6: **Research Contributions Overview**
Overview of how the different research contributions are linked together within this thesis.

- The proposed method was evaluated using a real-world tractor in a parking lot.

7. **A framework capable of supporting a separately trained external task with directed navigation and exploration** (Chapter 9)
   - A novel way to make learned RL exploration directed through integration with a separately trained task-specific module.
   - A novel representation based on Light Detection And Ranging (LiDAR) point clouds that is able to robustly bridge the sim2real gap between training in simulation and real-world usage.
   - A training approach capable of balancing directed and more generic exploration.
   - A warehouse simulator with procedurally generated warehouse layouts, that can be utilized to further build upon the presented work.
   - The proposed method was evaluated using a real-world Automated Guided Vehicle (AGV) in a warehouse environment.

In order to solve the question of how RL can be made more sample efficient an interplay between abstractions and adaptation will be required. Figure 1.6 illustrates how these various components are linked together.

1.6 List of Publications

The work presented in this thesis has also been disseminated in the following peer-reviewed publications:

Conference publications

- **Matthias Hutsebaut-Buysse**, Kevin Mets, and Steven Latré. *"Language grounded task-adaptation in reinforcement learning*."28th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, 2-4 October, 2020, Bruges, Belgium. 2020.
- **Matthias Hutsebaut-Buysse**, Kevin Mets, and Steven Latré. *"Pre-trained word embeddings for goal-conditional transfer learning in reinforcement learning."* Language in Reinforcement Learning Workshop at ICML 2020, the 37th International Conference on Machine Learning, 18 July, 2020, Vienna Austria.
- **Matthias Hutsebaut-Buysse**, Kevin Mets, Tom De Schepper, and Steven Latré. *"Disagreement Options: Task Adaptation Through Temporally Extended Actions."* Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, 2021.
- Abdellatif Bey Temsamani, Anil Kumar Chavali, Ward Vervoort, Tinne Tuytelaars, Gorjan Radevski, Hugo Van Hamme, Kevin Mets, **Matthias Hutsebaut-Buysse**, Tom De Schepper, Steven Latré. *"A multimodal AI approach for intuitively instructable autonomous systems: a case study of an autonomous off-highway vehicle."* The Eighteenth International Conference on Autonomic and Autonomous Systems, ICAS 2022, May 22-26, 2022, Venice, Italy.
- **Matthias Hutsebaut-Buysse**, Kevin Mets, Tom De Schepper, and Steven Latré. *"Structured Exploration Through Instruction Enhancement for Object Navigation."* The 34th Benelux Conference on Artificial Intelligence, BNAIC/Benelearn 2022, November 7-9, 2022, Mechelen, Belgium.
- Ferran Gebelli Guinjoan, Erwin Rademakers, Abdellatif Bey Temsamani, Gorjan Radevski, Tinne Tuytelaars, **Matthias Hutsebaut-Buysse**, Kevin Mets, Tom De Schepper, Steven Latré, Erik Mannens, and Hugo Van Hamme. *"A Multi-modal AI Approach For AGVs: A Case Study On Warehouse Automated Inventory."* The Nineteenth International Conference on Autonomic and Autonomous Systems, ICAS 2023, March 13-17, 2023, Barcelona, Spain.
- **Matthias Hutsebaut-Buysse**, Ferran Gebelli Guinjoan, Erwin Rademakers, Steven Latré, Abdellatif Bey Temsamani, Kevin Mets, Erik Mannens and

Tom De Schepper *Directed Real-World Learned Exploration* The 2023 IEEE/RSJ International Conference on Intelligent Robots, IROS 2023, October 1-5, 2023, Detroit, USA.

Journal publications

- **Matthias Hutsebaut-Buysse**, Kevin Mets, and Steven Latré. *"Hierarchical Reinforcement Learning: A Survey and Open Research Challenges."* Machine Learning and Knowledge Extraction 4.1 (2022): 172-221.
- Abdellatif Bey Temsamani, Ferran Gebellí Guinjoan, Erwin Rademakers, Anil Kumar Chavali, Ward Vervoort, Tinne Tuytelaars, Gorjan Radevski, Hugo Van Hamme, Kevin Mets, **Matthias Hutsebaut-Buysse**, Tom De Schepper, Steven Latré and Erik Mannens *"A Multi-modal AI Approach For Intuitively Instructable Autonomous Systems"* International Journal On Advances in Systems and Measurements 16.1 (2023): 1-13.

1.7 Outline

The next part of this thesis will introduce the necessary preliminaries. Chapter 2 first introduces the formal definition of sequential decision-making problems. This first part also includes Section 2.6, which includes a brief overview of the definition and state-of-the-art of common tasks in robotic navigation.

The second part of this thesis introduces the concept of abstraction in deep RL. Chapter 3 contains a survey of the different approaches that have been introduced in RL. This survey concludes by addressing open challenges. A novel framework for solving object navigation tasks through abstraction is introduced in Chapter 4.

The third part of this thesis focuses on the topic of task adaptation. Chapter 5 introduces an approach to performing task adaptation through sampling task adaptations. An alternative approach that replaces this sampling approach with one based on pre-trained word embeddings is introduced in Chapter 6. Finally, this method is extended in Chapter 7 with a method to balance exploiting prior knowledge and exploring a novel task.

The final part of this thesis revolves around applying RL in real-world applications. In Chapter 8 real-world PointGoal navigation is studied. In order to study differences between the real world and simulation, a novel office simulator is introduced in Section 8.2. Section 8.3 introduces a digital twin-based approach to carry out PointGoal navigation tasks in a real-world setting. In the final chapter of this thesis (Chapter 9) a method is introduced which is capable of exploring a warehouse environment in a directed manner.

2

Preliminaries

The navigation problems studied in this thesis can be formally described as sequential decision-making problems. In a sequential decision-making problem, earlier actions typically influence later available options.

2.1 Markov Decision Processes

A Markov Decision Process (MDP) (Howard, 1960; Puterman, 1994) is a formal model, used to describe discrete-time stochastic control processes. The MDP model is useful for representing decision-making problems in which an agent (the decision maker) can influence the process by executing actions.

An MDP is defined by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, in this tuple:

- $\mathcal{S}$ defines the set of possible states the agent can be in, also called the state space,
- $\mathcal{A}$ contains the set of actions the agent can execute, also called the action space,
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the probabilistic state transition function. This function determines what the outcome state of an action in a state will be.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, an agent can receive a certain reward when visiting certain states after executing an action.
- $\gamma \in [0, 1]$ is a discounting factor

The agent exercises control over the MDP by executing actions. A solution to an MDP can be expressed with a policy π. The policy models which action the agent takes in each state: $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The interaction trace $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$ between the agent and the environment is called a trajectory τ.

The objective in an optimal control problem is to find an optimal policy π^* that maximizes the return, which is defined as the expected accumulated discounted reward along a trajectory:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \gamma^t R\left(s_t, a_t, s_{t+1}\right) \right] \tag{2.1}$$

In order to assess the quality of a specific policy, the concept of a value function can be utilized. A value function $V_\pi(s_t)$ represents the total expected discounted reward starting in state s_t and following policy π:

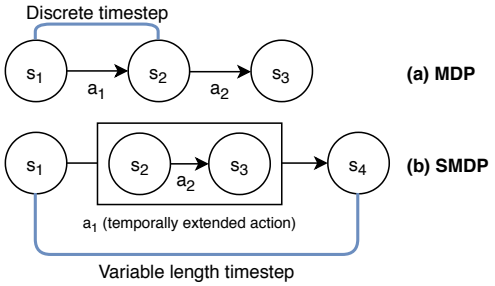
$$V_\pi(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^t R(s_{t+k}, a_{t+k}, s_{t+k+1}) \right] \tag{2.2}$$

There might be multiple policies that are optimal, but they all share a common value function V^*.

In an episodic MDP, the state of the MDP is reset after a finite number of T-steps. In a non-episodic MDP $T = \infty$ and the objective is often changed to learn a policy that maximizes reward on average, each timestep (Mahadevan, 1996).

In an MDP, the probability of future state transitions only depends on the current state and action and does not take the history of prior states into account. This assumption is called the Markov property. In practice, this entails that the current state should contain all the information an agent needs to make optimal action decisions. And thus the agent does not require the usage of a memory of past states.

The received reward is typically discounted using a discount factor γ. The reward received in the distant future is often considered less valuable to the agent than the reward we can receive immediately. If $\gamma < 1$ this also entails that cumulative rewards of trajectories are finite. This is especially important when the MDP is non-episodic and can go on forever.



Figuur 2.1: **MDP vs SMDP framework**
The MDP framework differs from an SMDP, in that an SMDP allows the length between timesteps to be variable. This is an essential property to support temporally extended actions.

2.1.1 Partially Observable Markov Decision Processes

However, assuming that the complete underlying state of the environment is available to the agent is in a lot of cases an unreasonable assumption. An agent often only has a partial, and noisy observation of the underlying state. For example, if a camera is utilized as input to the agent, the agent only has information about what is happening in front of the agent but does not have any information about what is behind the camera. Due to camera noise and artifacts, the agent might also only have a distorted observation of the state.

The setting in which the agent only has a partial view of the state is described using a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998). The MDP model is extended in order to describe a POMDP by introducing an additional observation-space Ω and a function $\mathcal{O}(s_t)$ which can be utilized to sample observations $o_t \sim \mathcal{O}(s_t)$. In a POMDP a policy consists of a mapping from the history of observations (the belief state) to an action.

Calculating exact solutions for a POMDP problem is only trackable for very simple problems, due to the large amounts of possible belief states. Approximation methods have however been proposed (Hauskrecht, 2000).

2.1.2 Semi-Markov Decision Processes

In the MDP framework an action is taken on each discrete time step. However, when working in a continuous time space, or when dealing with temporally extended sequences of actions (e.g., a sub-behavior that takes the agent to a different part of the state space), different actions might have different execution lengths, as demonstrated in Figure 2.1.

In order to support the variable duration of a single action, the MDP framework has been extended to the Semi-Markov Decision Process (SMDP) framework (Bradtke and Duff, 1995; Mahadevan et al., 1997; Parr and Russell, 1998b). This was done by adding a random variable T to the transition function $\mathcal{P}$. This random variable denotes the time between actions. In the case of an SMDP the reward-function $R(s_t, a_t, s_{t+1})$ denotes the cumulative reward collected during T steps after executing a_t in s_t.

2.2 Dynamic Programming

When provided with the transition function $\mathcal{P}$ and reward function $\mathcal{R}$, how could one obtain an optimal policy π^* in a finite MDP (containing a finite number of states and actions)? Both policy iteration and value iteration are two dynamic programming methods that can be utilized to find an optimal policy in this setting.

These methods make use of the Bellman equation (Bellman, 1952) which states that the value of a state is equal to the immediately received reward combined with the discounted value of the expected next state. This equation allows the problem to be broken down into smaller sub-problems.

2.2.1 Policy Iteration

Policy Iteration is a dynamic programming method that can be utilized to calculate an optimal policy. This method consists of two phases. In the first phase, the current policy π is evaluated by obtaining the value function by iterating over all states s and result states s':

$$V_\pi(s) \leftarrow \sum_{s',r} P\left(s', r \mid s, \pi(s)\right)\left[r + \gamma V\left(s'\right)\right] \tag{2.3}$$

This is done iteratively until the change between the old and new value function $|V_\pi^{new}(s) - V_\pi^{old}(s)|$ becomes smaller than a small positive number for each state.

In the second phase the policy is improved in each state by utilizing the obtained value function:

$$\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} P\left(s', r \mid s, a\right)\left[r + \gamma V\left(s'\right)\right] \tag{2.4}$$

This is also done repeatedly until the policy obtained becomes stable (which can be defined as outputting the same actions as the prior policy for each state). Unless the policy is already optimal, each step in this process is guaranteed to improve the policy.

2.2.2 Value Iteration

Value Iteration is an alternative approach in which the focus is on iterating the value function in order to obtain an approximation of the optimal value function V^*:

$$V_\pi^{new}(s) \leftarrow \max_a \sum_{s',r} P\left(s', r \mid s, a\right)\left[r + \gamma V_\pi^{old}\left(s'\right)\right] \tag{2.5}$$

The value function is iteratively updated until individual state values do not change anymore, and $|V_\pi^{old}(s) - V_\pi^{new}|$ becomes smaller than a small number.

Once the optimal value function is obtained an optimal policy π^* can be derived from it:

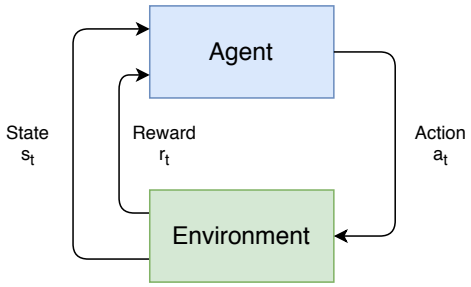
$$\pi^*(s) = \arg\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma V^*\left(s'\right)\right] \tag{2.6}$$

While value iteration is a considerably simpler algorithm to implement compared to policy iteration, it is also generally more expensive in terms of computational costs (Pashenkova et al., 1996).

2.3 Reinforcement Learning

Assuming full knowledge of the MDP, dynamic programming approaches can be utilized to obtain an optimal policy π^*. However, full knowledge of the underlying MDP is often an unrealistic assumption. In most realistic sequential decision-making processes, the transition function $\mathcal{P}$ and the reward function $\mathcal{R}$ are unknown to the agent. In such cases, we are not able to calculate the next state and reward from the current state and action. It might only be possible to obtain the next state by actually executing an action in the environment.

Additionally, if the number of states and actions is too large (or even infinite) it might not be possible to utilize dynamic programming, which typically requires multiple iterations over all states.



Figuur 2.2: **The reinforcement Learning Framework**

Reinforcement Learning (RL) (Sutton and Barto, 2018) is the problem set, concerned with handling unknown factors in the MDP, by utilizing a trial-and-error approach. In the model-free RL setting, the agent does not have access to the environment dynamics and does not try to explicitly learn such a model. In the case of model-based RL the agent is either provided with a model of the environment dynamics or learns an explicit predictive dynamics model.

An RL agent starts in an initial state $s_0 \in \mathcal{S}$. At each time step t the agent interacts with the environment by taking an action $a_t \in \mathcal{A}$. On each timestep, the agent can either act *greedy*, and exploit what it already has learned by following its current policy $\pi(s_t)$. Alternatively, the agent can also choose to explore the outcome of a different random action in order to possibly improve its policy. This distinctive interaction between agent and environment is pictured in Figure 2.2.

RL could for example be utilized for selecting your next restaurant reservation. You can either act greedy, and visit your current favorite place, or explore a new restaurant. When exploring the new place, you might end up not liking it, but it can of course also become your new favorite restaurant.

Balancing this *exploration-exploitation* dilemma is one of the characteristic problems of RL. Different exploration schemes exist. For example, by adding noise to the action-space ϵ-greedy takes random actions $\epsilon\%$ of the time. The ϵ parameter is often decayed over a number of steps. Other exploration schemes add noise directly to the policy parameters (Plappert et al., 2018; Fortunato et al., 2018), add an intrinsic curiosity bonus to the reward signal (Burda et al., 2018b,a; Bellemare et al., 2016; Houthooft et al., 2016), or use a distributional perspective (Dabney et al., 2018a; Bellemare et al., 2017).

After taking action a_t, the agent gets feedback in the form of a reward-signal r_{t+1}, and a new state s_{t+1}. The second key problem in RL, the credit assignment problem is concerned with figuring out which of the previously taken action (or set of actions) led to a delayed reward signal.

The reward an agent receives directly from the environment is called the extrinsic reward. Extrinsic, because it is external to the agent. These reward signals can be dense or sparse. An open field in which the agent gets the distance to the goal state after each action is an example of an environment with a dense reward signal. Alternatively, if the agent only receives feedback upon reaching the goal state, the reward is sparsely observed. An environment with a dense reward structure makes the credit assignment problem more tractable. However, in most realistic environments, non-zero rewards are often only sparsely observed.

Similar to the dynamic programming approaches of finding optimal policies directly (policy iteration) or indirectly through a value function (value iteration), RL approaches can also be split into value-based and policy gradient methods.

2.3.1 Value-based Methods

Value-based methods are an indirect RL approach. They utilize a value function $V_\pi(s)$ in order to derive a policy. A value function is capable of estimating the cumulative discounted future reward (the value) starting from state s_t and following policy π:

$$V_\pi(s_t) = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^t R(s_{t+k}, a_{t+k}, s_{t+k+1})\right] \tag{2.7}$$

In order to decide what action to take given the current state s_t, a state-action-value function (or Q-function) can be used. This Q-function represents the estimated cumulative future discounted reward of taking an action a_t in state s_t while following policy π:

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right] \tag{2.8}$$

The goal of the agent is to come as close as possible to the optimal Q-function. The optimal Q-function (Q^*), is capable of outputting the maximum achievable cumulative reward, starting in state s_t, and taking action a_t.

$$Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t) \tag{2.9}$$

This function can be learned through algorithms such as the off-policy Q-Learning (Watkins and Dayan, 1992) algorithm or the on-policy SARSA algorithm (Rummery and Niranjan, 1994). An on-policy algorithm updates the policy only with samples gathered by utilizing this same policy. Off-policy learning is typically more sample-efficient, as it also is capable of utilizing experiences gathered when

following a different policy (e.g., a random exploration policy) and re-using prior experiences gathered by utilizing previous versions of the policy.

Q-learning is an off-policy approach, because it does not use the current policy to estimate the total value of the next state s_{t+1}, but instead uses the highest expected obtainable value.

$$Q_\pi\left(s_t, a_t\right) \leftarrow Q_\pi\left(s_t, a_t\right) + \alpha(r_t + \gamma \max_a Q_\pi\left(s_{t+1}, a\right) - Q_\pi\left(s_t, a_t\right)) \quad (2.10)$$

While SARSA is considered on-policy because it uses the current policy to sample the next action a_{t+1}:

$$Q_\pi\left(s_t, a_t\right) \leftarrow Q_\pi\left(s_t, a_t\right) + \alpha(r_t + \gamma Q_\pi\left(s_{t+1}, a_{t+1}\right) - Q_\pi\left(s_t, a_t\right)) \quad (2.11)$$

In both algorithms, α is the learning rate used to gradually update the old function.

In order to actually learn the Q-function from interaction (s_t, a_t, r_t, s_{t+1}) with the environment, both Q-learning and SARSA make use of the Bellman equation (Bellman, 1952), which allows to recursively update Q-values:

$$Q_\pi(s_t, a_t) = \mathbb{E}\left[r_t + \gamma Q_\pi\left(s_{t+1}, a_{t+1}\right)\right] \quad (2.12)$$

Once the agent has iteratively learned a good estimate of Q^* from interaction with the environment, a greedy policy can be derived:

$$\pi^*(s_t) = \arg\max_a Q^*(s_t, a) \quad (2.13)$$

2.3.2 Policy Gradient Methods

An alternative family of policy gradient methods directly searches for a policy π_θ parameterized by θ. The parameters of the policy are updated through the gradient of an objective function $J(\theta)$ and a learning rate α:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t) \quad (2.14)$$

The objective function $J(\theta)$ that is being optimized in this setting can be defined as the total expected future value:

$$J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^k r_t\right] = V_{\pi_\theta} \quad (2.15)$$

Unfortunately calculating $\nabla J(\theta)$ is non-trivial because it depends on the stationary distribution of states:

$$J(\theta) = \sum_{s \in \mathcal{S}} d_\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q_{\pi_\theta}(s, a) \tag{2.16}$$

In order to obtain this distribution d_π one typically requires access to the (unknown) environment dynamics ($d_\pi \mathcal{P} = d_\pi$). The policy gradient theorem (Sutton et al., 2000) however allows the gradient to be represented without requiring $d_\pi(s)$:

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \Phi_t \right] \tag{2.17}$$

In practice, policy gradient algorithms try out different actions (policy evaluation) and increase the likelihood that the policy will sample successful actions which led to high future returns again, when in similar states (policy improvement).

In order to perform this policy improvement step, the agent needs an estimate of the total future value of an action in a state (the RL prediction problem). To obtain such value estimations, Monte Carlo simulation can be used to estimate the cumulative future discounted reward by utilizing entire past trajectories. This approach is utilized by the REINFORCE algorithm (Williams, 1992).

$$\Phi_t = \sum_{t=0}^{\infty} \gamma^t r_t \tag{2.18}$$

Instead of utilizing entire trajectories to estimate the return, Actor-Critic (Konda and Tsitsiklis, 2000) methods utilize a learned action-value function (the critic) in order to estimate the return:

$$\Phi_t = Q_{\pi_\theta}(s_t, a_t) \tag{2.19}$$

Because returns obtained by sampling trajectories are often subject to high variance, it is also common to subtract a baseline $b(s_t)$ from the estimated return, in order to obtain a lower update variance, while remaining unbiased:

$$\Phi_t = \sum_{t=0}^{\infty} \gamma^t r_t - b(s_t) \tag{2.20}$$

A simple baseline could exist of subtracting an average reward obtained over a number of episodes. An advantage function is also often used, using the value of

the state $V(s_t)$ as the baseline:

$$\Phi_t = A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{2.21}$$

In order to facilitate exploration within policy gradient methods, an entropy regularizer can be used which would make the policy as random as possible, while still maximizing the expected return.

2.4 Deep Learning

When the state and action space are limited and discrete, elements of an RL architecture (such as the policy, and/or value function), can be represented using a tabular representation. However, this method quickly becomes intractable for more complex problems, which require the processing of high-dimensional input data such as images. Various function approximation methods have been used in the past in order to represent the different components of an RL agent. Example function approximation techniques include: linear models that learn through utilizing high-level expert-provided features of the state space (Diuk et al., 2008), and tree-based algorithms (Damien et al., 2005). However, more recently deep learning has become the most common approach to handling high-dimensional inputs in RL.

Deep neural networks have demonstrated to be capable of learning useful hierarchies of task-specific representations from raw high-dimensional input signals (LeCun et al., 2015; Schmidhuber, 2015). These representations can be used to perform complex machine-learning tasks in an end-to-end fashion. They are considered end-to-end because they are able to provide answers by using raw high-dimensional data directly, without the requirement of any preprocessing. Example tasks include: image classification (Dosovitskiy et al., 2021; Szegedy et al., 2016; He et al., 2016; Szegedy et al., 2015; Krizhevsky et al., 2012), image captioning (Wang et al., 2022; Karpathy and Fei-Fei, 2015), visual question answering (Driess et al., 2023; Antol et al., 2015), image generation (Rombach et al., 2022; Donahue and Simonyan, 2019; Radford et al., 2016; Gatys et al., 2015), sound generation (van den Oord et al., 2016), object detection (Redmon et al., 2016; Girshick, 2015), speech recognition (Schneider et al., 2019; Bahdanau et al., 2016; Sainath et al., 2013; Hinton et al., 2012), natural language translation (van den Oord et al., 2016; Sutskever et al., 2014), and natural language understanding (Ouyang et al., 2022; Devlin et al., 2019; Radford et al., 2019). These techniques have to some extent already been adopted in real-life practice such as in agriculture (Kamilaris and Prenafeta-Boldú, 2018) and medicine (Litjens et al., 2017).

2.4.1 Computer Vision

The progress made in the area of deep learning has been made mainly possible by a few building blocks. For example, in computer vision applications, convolutional neural networks (LeCun et al., 1998) have been demonstrated to be capable of capturing task-specific spatial dependencies. In such a network each layer uses learned filters in order to extract high-level features from the input in order to provide a lower dimensional representation as the original input to the downstream task. More recently, transformer-based approaches (Dosovitskiy et al., 2021) have become an alternative popular approach.

2.4.2 Sequential Data

Building blocks such as Long Short-term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) (Cho et al., 2014) recurrent networks allow deep learning to work with data which is sequential in nature. To accomplish this, they use gates that are capable of learning which parts of the sequential input data, summarized in a hidden state, should be passed on, and what to forget. These techniques are especially interesting when working with natural language, as the meaning of a word in a sentence is often dependent on other neighboring words. However, in RL too, memory can be useful (Hausknecht and Stone, 2015; Mnih et al., 2016), or even required, when the task is formulated as an POMDP.

However, when sequences become longer, LSTM and GRU-based approaches struggle due to their reliance on a single fixed-length vector to represent a long sequence of previous inputs. Attention (Bahdanau et al., 2015) is a mechanism introduced to solve this problem. When using Attention, instead of relying on a single hidden state, representing past members of a sequence, attention mechanisms learn how to weigh different hidden states associated with different past sequence members. More recently *transformer* architectures (Vaswani et al., 2017) solely rely on this attention mechanism and eliminate the usage of recurrent networks. The usage of a transformer architecture also typically allows for faster inference, as inputs can be processed in parallel.

2.4.3 Generative Approaches

Also, semi-supervised generative deep learning approaches, capable of generating new previously unseen data instances have recently achieved tremendous successes, and have been added to the standard set of deep learning building blocks. These techniques include Variational Autoencoders (VAE) (Kingma and Welling,

2014), which utilize a regularized bottleneck autoencoder approach. A regular autoencoder (Rumelhart et al., 1985) encodes a high-dimensional input in a lower-dimensional latent space. The VAE regularization tries to make sure that points close in the latent space are also similar once decoded back into their high-dimensional form. This regularization allows the generation of new data instances by generating points in the latent space.

Generative Adversarial Networks (GAN) (Goodfellow et al., 2014) generate new data using a generator network, which receives feedback that the generated instances look realistic or not by a second discriminator network. This approach has been demonstrated to be capable of generating out-of-sample high-dimensional images (Donahue and Simonyan, 2019) indistinguishable from real photographs.

2.5 Deep Reinforcement Learning

One of the key challenges of RL applied on problems with large state spaces is to be able to generalize feedback received from the environment to unvisited states, and untested actions. Using deep neural networks we can learn representations from raw high-dimensional inputs, which in turn can be used to approximate value functions, or directly express a policy. Unfortunately, combining non-linear function approximation, and iterative value function bootstrapping has been found prone to unstable learning (van Hasselt et al., 2018; Tsitsiklis and Van Roy, 1996), especially in an off-policy setting in which the policy used to obtain prior experiences might significantly differ from the current version of the policy. This problem has been identified as the *deadly triad* of RL (Sutton and Barto, 2018).

Combining RL with recent advancements in the area of deep learning has had a big impact on RL, giving birth to a new subfield called deep reinforcement learning (François-Lavet et al., 2018; Arulkumaran et al., 2017; Li, 2017; Mnih et al., 2015). Deep RL applies RL techniques, using high-dimensional state spaces, such as images (Justesen et al., 2019) or natural language (Luketina et al., 2019). This has been made possible due to the capability of deep learning algorithms to introduce different learnable layers of abstractions on the raw input data.

The seminal work of Mnih et al. (2015) demonstrated above human-level performance on a set of classic Atari 2600 video games (Bellemare et al., 2013). The introduced off-policy Deep Q-Network (DQN) architecture was able to learn different policies for a range of different video games, using only the screen pixels as input data. In order to generalize collected experience to unseen states, and untested actions, the $Q(s_t, a_t)$ function is represented using a deep neural network. The issue of instability caused by function approximation in RL (Tsitsiklis and Van Roy, 1996) was tackled by the usage of a separate target network, used to predict next-state future values. This target network is only periodically updated.

Additionally, an experience replay buffer (Zhang and Sutton, 2017; Lin, 1992) was used, in order to reduce temporal correlation in the observed state-reward sequences.

The DQN algorithm has been heavily studied, and various improvements have been proposed such as Double Deep Q-Networks (DDQN) (van Hasselt et al., 2016), prioritized replay (Schaul et al., 2016), dueling networks (Wang et al., 2016), multi-step learning (Sutton, 1988) and distributional RL (Dabney et al., 2018a,b; Bellemare et al., 2017). The Rainbow framework (Hessel et al., 2018) combines these improvements. While the original DQN-architecture could not achieve above-human performance on all tested games, the Agent57 approach (Badia et al., 2020), outperforms humans in all proposed test games, by automatically parameterizing a family of policies.

Various algorithms that directly optimize a parameterized policy such as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2016), Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017) have also been able to work directly on high-dimensional input spaces using deep neural networks. In order to stabilize learning, policy improvements are often restricted in order to avoid making too big updates, which could collapse performance.

Recent successes of deep RL include beating top professional human players in the complex board game of Go (Silver et al., 2016), and achieving human-level performance in cooperative 3D multiplayer video games such as DOTA 2 (OpenAI, 2019a, 2018), StarCraft II (Vinyals et al., 2019) and a modified version of Quake III Arena (Jaderberg et al., 2019).

2.5.1 Towards Real-world Deep Reinforcement Learning

There has also been significant progress towards applying deep RL in real-world applications such as robotics (OpenAI, 2019b; Mahmood et al., 2018; Kalashnikov et al., 2018; Levine et al., 2016a,b; Kober et al., 2013), dialog management systems (Cuayáhuitl et al., 2010), education (Mandel et al., 2014), autonomous vehicles (Bellemare et al., 2020; Pan et al., 2017; Ng et al., 2003) smart grids (François-Lavet et al., 2016), fleet management (Lin et al., 2018), resource management (Mirhoseini et al., 2018; Mao et al., 2016), nuclear fusion (Degrave et al., 2022) and recommender systems (Zhang et al., 2019). However, building real-world RL applications remains challenging, because RL algorithms often struggle with being sample efficient (Lee et al., 2019; Yu, 2018; Wang et al., 2017b; Strehl et al., 2006), that is, being able to learn a satisfying behavior from a limited amount of interaction with the environment. Furthermore, real-world RL systems require hard constraints in order to not damage equipment and allow safe exploration (Amodei

et al., 2016; García et al., 2015). Many practical issues (Zhu et al., 2020) need to be addressed as well, such as, how to reset the environment (Lu et al., 2020), and how to provide the agent with a reward signal. Real-world RL systems should also commonly be able to handle real-time systems, allowing states and actions to evolve simultaneously (Wang et al., 2020; Ramstedt and Pal, 2019; Doya, 2000) as the environment might have changed while the agent was selecting its next action.

2.6 Navigation

Navigation is at the core of Embodied AI (EAI), and the main focus of this thesis. Navigation tasks can be defined in a variety of different manners depending on how the goal is specified (e.g., a set of coordinates, or an object to navigate towards).

What is however a common assumption in these tasks, is that the agent only has access to egocentric observations of the environment. Typically, in the form of an RGB camera, depth camera, or a combination of both. The agent does not have access to a map of the environment. This is much like how humans are able to operate with only limited information.

The currently most studied tasks and approaches related to the work presented in this thesis are outlined in the following subsections.

2.6.1 Classical Navigation in Mobile Robotics

Autonomous mobile robots which are able to make decisions on what action to take based on their own perception of the world are starting to emerge in our everyday lives, and are increasingly more common in the industry.

Most mobile platforms encountered today (Rubio et al., 2019) however do not make use of RL, and typically do not operate in an end-to-end fashion. Different modules for locomotion, perception, cognition, and navigation (Siegwart et al., 2011) are often developed, and the output from one module is utilized by others. This approach is also often described as *Sense-Plan-Act*.

The locomotion modules allow the robot to actually move in the physical space. These modules most often depend on control theory and the understanding of various kinematics and dynamics models.

In order to obtain an observation of the state of the environment, a perception module is tasked with processing various signals obtained from various sensors. A laser scanner is for example able to determine the distance to various obstacles

placed near the robot. Sensors are often only accurate up to a certain point. Additionally, most sensors are not ubiquitously usable and may have trouble handling reflective surfaces or bad weather. Computer vision approaches, both traditional and deep learning based, are also heavily used when working with cameras.

The cognition module is the module that is responsible for the actual execution of the overall task. It does this by taking into account the input from the perception module and decides which action aligns best with the goal(s) of the agent.

An action of the cognition module could for example consist of navigating to a certain location. Such an action is typically not directly executed by the low-level control systems but handled by a separate navigation model which takes the state of the environment into account in order to devise a low-level navigation plan.

Traditionally navigation has been considered a geometric mapping and planning problem. Typical approaches try to build an accurate explicit representation of the world (a map). This map can then in turn be utilized to carefully calculate a solution through planning. If the environment is static, the agent can be provided upfront with such a map. The domain of Simultaneous Localization and Mapping (SLAM) (Smith and Cheeseman, 1986; Bresson et al., 2017) however is capable of dynamically constructing a map of the environment by moving around in the environment. In order to plan within a map the agent needs to be aware of its current location. Localization is often done through the usage of static beacons placed in the environment. SLAM approaches try to match past and current observations in order to relatively localize the agent within the map.

This modular approach has been widely adopted in controlled industrial static environments. In this setting, expensive sensors, guiding tracks, and beacon configurations are often worth the large investments. However, when applied using cheaper sensors and when being confronted with less static and predictable environments the robustness of these traditional approaches can become an issue (Lin et al., 2021; Bujanca et al., 2021; Mishkin et al., 2019). Due to the nature of a modular approach, making an error in one module will also influence the outputs of the other modules.

Additionally, individual modules are often rule-based. These rules are mainly designed by human experts but might contain mistakes, or they might not be able to handle unexpected situations.

2.6.2 Embodied AI Tasks

EAI offers an alternative to the modular Sense-Plan-Act paradigm. Instead of relying on chains of hand-crafted modules, EAI aims at solving tasks mainly in an end-to-end fashion through trial-and-error learning. The currently most actively researched tasks and their main approaches are presented in the following subsections.

2.6.2.1 PointGoal Navigation

In *PointGoal* navigation (Anderson et al., 2018) the goal of the agent is specified as a set of (relative) coordinates. The agent is successful if it is able to utilize a designated *done* action close to the specified set of goal coordinates. This is a trivial task if the environment is completely empty, but difficult if the environment is filled with obstacles.

When the agent has access to a perfect GPS+Compass sensor, this task has been considered solved in noiseless simulated photorealistic environments through utilizing a near-linear scalable distributed approach (Wijmans et al., 2020). This DD-PPO approach allowed the agent to interact 2.5B times with the environment, which would take a human about 80 years. However, interestingly 90% of its performance is achieved in the first 100M interactions. This indicates that the PointNav task is relatively easy to get a rough policy trained on, however, it is very hard to further fine-tune.

As this is a large amount of compute, efforts (Wijmans et al., 2022a) have been made to reduce the required number of computing resources by fine-tuning the DD-PPO approach. A different approach in order to improve the sample efficiency in PointNav was done through the introduction of auxiliary tasks (Ye et al., 2020; Desai and Lee, 2021). Examples of such auxiliary tasks include the prediction of the action taken given two observations or the prediction of the distance between two observations taken from a prior trajectory.

The visual encoder part of most of the introduced approaches currently relies on ResNet-based architectures. Alternative visual backbones have also been studied. A CLIP-based (Khandelwal et al., 2022) solution has been found to be more successful at encoding primitives than ImageNet-pre-trained visual representations.

Unfortunately, in a more realistic setting (pointnav-v2) in which the agent has no access to a GPS+Compass sensor, and observations and movements are noisy the performance quickly drops. Zhao et al. (2021) proposed to replace the GPS+Compass sensor with a learned visual odometry model. By doing so a Success Rate (SR) of 71.7% was achieved on the Habitat 2021 challenge (Kadian et al., 2020). Partsey et al. (2022) verified that localization, and not sensor/actuation noise is the actual bottleneck in the more realistic (pointnav-v2) setting. They continued to build upon the approach of using a learned visual odometry module. Through some extensions (action embeddings, data augmentations, larger datasets, and model sizes) of this module, they were able to improve performance to 96% SR and 77% Shortest-Path Length (SPL).

Because high computational demands remain a limiting factor, further efforts have been made (Wijmans et al., 2022b) to further increase the scaling of RL algorithms on this task by reducing the required synchronization points in on-policy learning.

Dataset	Approach	Success Rate	SPL
Gibson	Random (Savva et al., 2019)	0.03	0.02
Gibson	Blind (Savva et al., 2019)	0.62	0.42
Gibson	DD-PPO (Wijmans et al., 2020)	0.996	0.941
MP3D	Random (Savva et al., 2019)	0	0
MP3D	Blind (Savva et al., 2019)	0.35	0.25
MP3D	DD-PPO (Ramakrishnan et al., 2021; Wijmans et al., 2020)	0.94	0.83
HM3D	DD-PPO (Ramakrishnan et al., 2021; Wijmans et al., 2020)	0.99	0.92

Tabel 2.1: SOTA performance on the PointNav-v1 task (using a depth sensor). In the v1 task, the agent has access to a compass+GPS sensor

Dataset	Approach	Success Rate	SPL
Gibson	Visual Odom (Zhao et al., 2021)	0.82	0.63
HM3D	Visual Odom (Zhao et al., 2021)	0.94	0.74

Tabel 2.2: SOTA performance on the PointNav-v2 task. In the v2 task, the agent has no access to a compass+GPS sensor

PointGoal navigation can be considered the most well-understood task in EAI (Table 2.1 and 2.2). The research conducted on this task has also led to an interesting insight that navigation does not require a map. This is in contrast to how navigation is typically solved in more traditional robotic setups through the usage of various SLAM techniques in which a map is constructed based on egocentric observations of the agent. This map in turn is then utilized to plan trajectories.

2.6.2.2 ImageGoal Navigation

In the ImageGoal task (Zhu et al., 2017) the agent receives an image of a goal location in the environment. The agent is deemed successful if it manages to find the location in which the goal image was taken. This task is especially interesting because the modality of the goal and the observation are the same (e.g., an RGB image).

Zhu et al. (2017) proposed a deep siamese model which utilizes a shared visual processing for the goal image and the current observation. This fused state and goal embedding is then provided to scene-specific layers which should encode specific characteristics of the current environment such as the floor plan and object arrangements.

A different approach that only utilizes passive pre-recorded videos has also been applied in this setting (Hahn et al., 2021). In this approach, a topological map is maintained and a learned visual and semantic module is utilized to sample subgoals, which are defined by a distance and angle. These subgoals can then be reached through a simple visual planner.

2.6.2.3 ObjectGoal Navigation

The ObjectGoal task (Anderson et al., 2018) differs from the PointGoal task in the sense that the agent receives an object category instead of a set of coordinates as its goal input. The agent is considered successful if it manages to navigate close to an instance of the specified object category and utilizes a special *done*-action. In order to solve this task, the agent will need to learn about what the goal object actually looks like (visual grounding), and make active exploration decisions about where the agent should actually be looking for such an object.

Scaling end-to-end approaches, which rely on massive amounts of interactions, has only been proven limited successful in this task reaching a SR of about 13% (Wijmans et al., 2020, 2022b). Furthermore, it was found that purely end-to-end learning approaches can greatly overfit datasets (Maksymets et al., 2021). Datasets typically contain only a limited amount of static goal objects, the introduction of additional goals in random positions demonstrated to somewhat reduce this overfitting issue.

Early attempts at cracking the objectnav task resolved around introducing episodic semantic maps (Chaplot et al., 2020a) based on a pre-trained object segmentation and classification module. Learning to construct a semantic map of a novel environment allows the agent to exploit semantic priors of the relative arrangement of previously discovered objects. A different module is then often utilized to select a long-term goal in the form of a point to reach within the build map. The task of then actually navigating to the proposed point can be either performed by a classic planning approach or a policy trained using RL (Chaplot et al., 2020b). Chaplot et al. (2021) further improved this approach by utilizing the uncertainty of the pre-trained object detector as a reward signal for a policy utilized for collecting novel training data for the perception module.

A different end-to-end approach consists of utilizing auxiliary tasks including map coverage prediction, inverse dynamics models and how much of the goal object is visible in an observation (Ye et al., 2021), together with an exploration policy, which is trained to maximize floor coverage. The initial exploration policy is utilized to bootstrap training. However, as this tends to lead to over-exploration, in a later stage of training this policy is not used anymore.

In computer vision and natural language processing tasks, the availability of large datasets has proven to be instrumental in training complex models. However, datasets utilized in ObjectNav research typically depend on scans of real-world buildings. Creating and annotating such scans is a costly, and time-consulting enterprise, posing practical limitations on the scale of many datasets. Recently approaches (Deitke et al., 2022) which make use of procedurally generated environments for pre-training have improved the performance on many baseline tasks including ObjectNav.

Dataset	Approach	Success Rate	SPL
MP3D (21 objects)	Random	0	0
MP3D	Human (Ramrakhya et al., 2022)	0.94	0.43
MP3D (21 objects)	DD-PPO (E2E baseline)	0.06	0.02
MP3D (21 objects)	THDA (Maksymets et al., 2021)	0.20	0.09
HM3D (6 objects)	ByteBOT (Ruiz and Todt, 2021)	0.68	0.37
Gibson	PONI (Ramakrishnan et al., 2022)	0.74	0.41

Tabel 2.3: SOTA performance on the ObjectNav task

A competitive approach based on Imitation Learning (IL) has also recently been proposed. The Habitat-Web dataset (Ramrakhya et al., 2022) contains 70k human demonstrations. Utilizing these examples allowed an agent to pick up sophisticated human-like exploration behaviors such as peeking around corners or scanning a room to get a sense of its contents. Ramakrishnan et al. (2022) proposed to utilize a supervised learning-based approach centered around the question of where to look for a specific object type in order to predict long-term goal locations in the semantic map introduced in Chaplot et al. (2020a).

Table 2.3 gives an overview of some of the mentioned methods and their performance on ObjectNav tasks.

2.6.2.4 AreaGoal Navigation

AreaGoal navigation (Anderson et al., 2018; Wu et al., 2018) challenges the agent to visit a specific region in the environment. In domestic settings, these regions are often rooms.

Narasimhan et al. (2020) proposed to tackle this task by utilizing a learning-based system that is capable of outputting a semantic belief map of the current layout. Through utilizing learned stylistic regularities in houses, this approach is able to predict the locations of rooms the agent has not yet seen. Through exploring the environment, the agent is capable of updating its beliefs about the layout of the environment.

2.6.2.5 Question Answering

An overarching task in EAI is the task of Embodied Question Answering (EQA) (Das et al., 2018a). In this setting, the agent receives a question formulated in natural language and is tasked with finding an answer. In order to find an answer the agent needs to interact with the environment.

PACMAN (Das et al., 2018a) is a hierarchical approach in which the high-level

planner selects an action (e.g., move forward), and the low-level controller decides how many times the action should be executed (e.g., until the end of a corridor is reached). If the planner selects a stop action, a separate question-answering module is utilized to answer the question. This approach was trained using IL and fine-tuned using RL. Adding RL fine-tuning improved the amount of successfully answered questions. However, consistent with other research (Ye et al., 2021) it was found that by utilizing RL the agent also often overshoots the target due to over-exploration.

Das et al. (2018b) further developed the utilization of abstractions in the EQA task. These abstractions include a separately trained objectnav module and an area-goal module.

2.6.2.6 Rearrangement

The release of the second version of the Habitat simulator (Szot et al., 2021) also included the definition of a set of novel benchmarking tasks for EAI. In these tasks, the agent is required to rearrange the environment. These tasks are proposed to work towards a mobile robot equipped with a manipulator which could act as an intelligent home assistant. This assistant could help out with tasks such as tidying the house, preparing groceries, or setting the dinner table.

In the initial baseline experiments, it was found that current flat RL approaches are not suitable for solving this kind of task. One of the reasons this was found to be challenging can be attributed to the complexity of designing reward functions capable of providing the agent feedback about a set of different sub-tasks such as navigating, picking up objects, or placing objects in different locations.

A solution based on abstraction (different sub-skills) was proposed (Szot et al., 2021) as these can be trained using relatively simpler individual reward functions. However, in this hierarchical approach navigation remains challenging, as the agent is not provided with the position it should navigate to, but with the position of a target object. The agent thus also needs to figure out which position to navigate to in order to successfully pick or place a target object. Additionally, hand-off problems were observed in which the execution of a skill would hinder the execution of another (e.g., not close enough positioned to a target object, accidentally closing a drawer, or knocking over an object).

Further research found that scaling training (Wijmans et al., 2022b) solved these issues. When not limiting navigation actions from manipulator skills (pick and place) the pick and place skills also learned to navigate early during training, thus allowing them to correct their position when placed too far from the target object due to a prior skill execution.

Gu et al. (2022) proposed a similar approach in which all skills are allowed to navigate. It was also demonstrated that specifying goals as regions instead of points for the navigation skill proved more effective. This approach was the winner of the first Habitat Rearrangement challenge (Szot et al., 2022) by achieving a SR of 64% on the easy track, and 43% SR on the regular track. In the easy track all containers (e.g., drawers and a fridge) are starting open, in the regular track the object to be rearranged could be in a closed container.

The Habitat-Web dataset (Ramrakhya et al., 2022) contains 12k human demonstrations of pick and placement tasks. Through the usage of IL a performance of 17.5% SR (9.8% SPL) on new object-receptacle initializations was achieved. Training a policy using the same approach but with calculated shortest path examples the agent only achieves a SR of 1.9% and SPL of 1.8%. These results indicate that also in this task mimicking human behavior can also assist an agent in generalization to unseen settings.

2.6.3 Real-world Learned Navigation

In order to utilize learning-based approaches in real-world settings multiple approaches are currently being actively researched (Levine and Shah, 2023):

A first option consists of deploying a traditional navigation stack, but relying on machine learning (e.g., computer vision) in order to handle the semantic aspects. An example of such an approach could consist in utilizing a frontier-based exploration method (Yamauchi, 1997) in order to navigate, and stop the agent once a vision system detects the object it was searching for.

Learning-based approaches have also been introduced in the mapping problem (Chaplot et al., 2020a). The learned map can then in turn be utilized in order to apply geometric planning methods.

However, either assisting in the semantic aspect or the mapping aspect of a classic navigation approach has its limitations. For example, muddy terrains might be mapped as navigable on a map, although in practice a robotic platform might not be able to navigate through the mud. An embodied approach would consist of directly interacting with the world and learning implicit environmental representations obtained directly from sensor inputs. When learning is successful these representations can go beyond simple occupancy maps but will provide real information about affordances and traversability.

Unfortunately, directly applying trial-and-error learning in a real-world environment is often not possible, due to the large amount of required interactions. When a simulated version of the environment is available a sim2real approach can be an alternative option. In this setting, the embodied RL agent can directly be trained

in an end-to-end fashion in simulation. If the difference between simulated and real-world sensors is not too large, the resulting policy can be directly utilized in the real world. If however, the sim2real gap is too large techniques relying on domain randomization (James et al., 2017; Tobin et al., 2017) and domain adaptation (Patel et al., 2015; James et al., 2019; Rao et al., 2020) can be utilized. A major issue when relying on a simulator is that one must also take various edge cases into account. For example, if the desired agent should be capable of navigating in areas with a lot of highly reflective surfaces such as glass, the simulator should also have an accurate representation of these surfaces. When in a purely geometric planning approach one would build a module capable of handling reflective surfaces, the engineering effort is now shifted towards designing the simulator.

Directly utilizing end-to-end trial-and-error-based learning in the real world would overcome this issue. However, the required amount of interactions makes this still an impractical approach.

Alternatively, real-world (human) demonstrations could be obtained and utilized in order to train the policy directly on real-world sensors and dynamics. This can be done through utilizing imitation learning (Pomerleau, 1988; Bansal et al., 2018; Codevilla et al., 2019), directly learning a mapping from states to actions, or through offline RL (Levine et al., 2020), which offers various forms of explicit regularization in order to prevent out-of-distribution actions.

3

Hierarchical Reinforcement Learning

The contributions presented in this chapter are based on the publication titled: *"Hierarchical Reinforcement Learning: A Survey and Open Research Challenges"*.

3.1 Introduction

As discussed in the previous chapters, RL has been proven to be a powerful method for solving sequential decision-making problems. RL agents are capable of learning how to solve a problem from interactions with its environment. In order to solve the problem, the agent does not need to know the dynamics of the environment in advance. A successful RL system will efficiently utilize experience gathered during trial-and-error learning, in order to maximize the reward signal.

Hierarchical Reinforcement Learning (HRL) extends the capabilities of RL, by introducing a divide-and-conquer approach. In this approach, the complex, difficult-to-solve problem, is abstracted into multiple smaller problems. These abstracted problems are generally easier to solve and their solutions can be reused to solve different problems. This approach has previously been successfully utilized (Georgievski and Aiello, 2015; Dean and Lin, 1995; Sacerdoti, 1973) to speed up many offline planning algorithms where the dynamics of the environment are known in

advance.

This compositionality and the usage of abstractions have been identified (McCarthy et al., 1955; Lake et al., 2017; Sutton and Barto, 2018) as one of the key building blocks of AI. Humans intuitively harness compositionality in order to tackle complex problems (Botvinick et al., 2009; Eckstein and Collins, 2019). For example, planning a vacation can be a complex endeavor if considered as a whole. However, when decomposed into multiple smaller tasks (e.g., selecting a hotel, booking a flight, arranging transportation to the airport) the task becomes more manageable.

Compositionality can be seen as a way to facilitate lifelong learning (Silver et al., 2013), learning new concepts, by combining previously learned primitives (Barreto et al., 2019). In addition, the power of compositionality to come up with, seemingly unlimited new concepts, based on meaningful primitives cannot be underestimated.

HRL aims to achieve compositionality by learning reusable sub-behaviors together with a composition strategy. While compositionality in HRL can be achieved in low-dimensional state spaces, deep neural networks (Section 2.4) have been demonstrated to be capable of automatically discovering composable representations, which offer significant opportunities for HRL, to facilitate compositionality directly using high-dimensional inputs.

Efficiently using such abstractions has proven to make significant contributions towards solving various important open RL problems such as reward-function specification (Kulkarni et al., 2016a), exploration (Jinnai et al., 2020), sample efficiency (Nachum et al., 2018b), transfer learning (Bacon et al., 2017), lifelong learning (Tessler et al., 2017) and interpretability (Beyret et al., 2019).

As previously discussed, an essential part of RL algorithms is that they use feedback in order to learn what is good and bad behavior. When feedback in the form of a reward signal is abundantly available, an agent can learn quickly. Unfortunately, specifying such dense reward signals is a complex challenge (Ng et al., 1999), and often results in unwanted side effects. An often-seen example of such a side effect in EAI tasks is over-exploration (Das et al., 2018a; Ye et al., 2021). Moreover, most control problems naturally come with a sparse reward signal (e.g., object grasped, destination reached). A sparse reward formulation makes learning extremely challenging (Ocana et al., 2023) as there is mostly only information on *what does not work*. HRL often utilizes various forms of intrinsic motivation (Schmidhuber, 2010) in order to provide additional denser reward signals for individual abstractions.

A second open challenge in RL we previously discussed, has been how to explore the environment efficiently (Burda et al., 2018b,a; Pathak et al., 2017; Bellemare et al., 2016). Recent empirical research (Nachum et al., 2019) demonstrated that

even simple forms of temporally correlating actions improves exploration efficiency. In this research, temporally extended exploration is seen as one of the most important contributions of HRL.

RL systems are also notoriously known for their sample inefficiency. While efficient use of abstractions seems a promising solution to this long-standing challenge, increased sample efficiency through HRL is most commonly only realized when amortizing computation over multiple iterations of very similar problems. Off-policy learning (Haarnoja et al., 2018b; Wang et al., 2017b; Gu et al., 2017) is a popular approach towards making RL more sample efficient. HRL algorithms unfortunately typically still require a more stable on-policy approach.

A third limitation of RL we target in this thesis consists of the focus of RL algorithms on solving only a single problem from scratch each time. However, abstractions that can be re-used and adapted when solving different tasks, pose a possible answer to the question of transfer learning in RL (Taylor and Stone, 2011). Drawing inspiration from research done in deep learning on how to transfer visual representations from one task to another (Yosinski et al., 2014; Bengio, 2012), HRL methods have shown to be capable of learning transferable abstractions within the same problem setting (Hausman et al., 2018). However, how to transfer abstractions between very different problems remains an open problem.

While the potential of HRL is vast, automatically discovering abstractions is non-trivial and remains an open research question. A lot of progress has been made recently, utilizing various forms of temporal and state abstractions. However, solutions currently either heavily depend on expert knowledge, do not scale well, or are sample inefficient.

The goal of this chapter is to provide a comprehensive understanding of how various HRL algorithms contribute towards solving the above-described open challenges in RL. While initial steps in HRL have been previously surveyed (Barto and Mahadevan, 2003), the capability of deep learning to work with high dimensional data has inspired a whole new set of directions and possibilities in HRL. For example, the challenge of end-to-end discovering, and sequencing temporally extended actions, directly from high-dimensional inputs, is a novel direction that has become possible because of recent breakthroughs in deep learning. While also briefly covering early research, the focus of this chapter is on these new directions inspired by deep learning.

The major part of this chapter consists of a survey of three frameworks and their most common implementations (problem-specific models in Section 3.5, options in Section 3.6, and the goal-conditional framework in Section 3.7). An overview of benchmark environments and tasks used in HRL is provided in Section 3.8. An evaluation of the discussed frameworks and algorithms is displayed in Section 3.9. In order to spark future research, a list of potential research directions is provided

in Section 3.10, which inspired the novel approaches outlined in this thesis.

3.2 Abstraction Mechanisms

One of the core mechanisms of HRL is the usage of temporal abstractions. Solving complex problems often involves reasoning on a hierarchy of multiple time scales. For example, when driving to work you typically do not start planning about whether you should steer left or right. You start on a much higher level, by planning for example what roads to take, and whether you should stop to refuel.

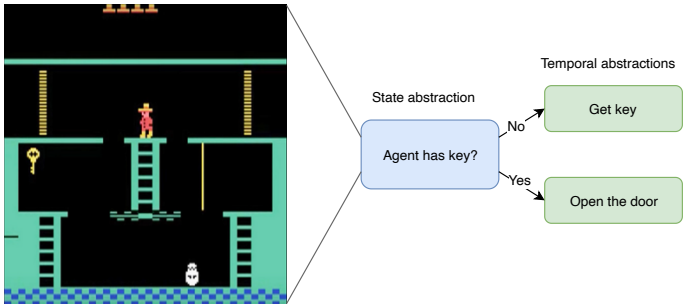
A temporally extended action (sub-behavior), consists of a sequence of primitive actions and possibly other temporally extended actions. These temporal abstractions can be utilized by a learning agent in order to make decisions on a higher level of abstraction. The sequence of actions that make up a temporally extended action can be fixed, or governed by a policy.

A second form of abstraction that is common in HRL are abstractions imposed on the state space (Li et al., 2006). It is not feasible to learn the best action in every possible state in a high-dimensional input space. Instead, we use state abstractions (e.g., enemy visible on screen?), to reason about what action to take. Appropriate state abstractions have been demonstrated to significantly increase learning performance in RL problems (Konidaris and Barto, 2009a; Diuk et al., 2008).

For example, similar states in terms of transition dynamics and reward function, measured using the notion of bisimulation (Larsen and Skou, 1991) for MDPs, can be grouped to build state abstractions (Ferns et al., 2004). These can be used to transfer an optimal policy from one MDP to a larger one (Castro and Precup, 2010) and to discover temporal abstractions (Castro and Precup, 2012).

Current deep RL algorithms are capable of learning their own state abstractions from raw high-dimensional state spaces. These algorithms are for example capable of detecting doors, and learn that it is beneficial to walk through these doors. Unfortunately, these learned state representations are often not sufficiently expressive to reason about complex environments and develop long-horizon plans. Such a plan might for example be to search for a key and return to the location of the key when confronted with a locked door.

Given abstractions over the state space, and linked temporally extended actions, the agent can reason on a higher level about the decisions it can make. Instead of learning a single behavior for controlling the entire raw state space, a hierarchy of different systems is able to control various abstracted parts of the state space through temporally extended actions. An example of such a hierarchy is presented in Figure 3.1.



Figuur 3.1: **An example of the usage of state- and temporal- abstractions in the *Montezuma's Revenge* Atari 2600 video game**
The agent uses a state abstraction (agent has a key), in order to decide upon which temporally extended action to activate.

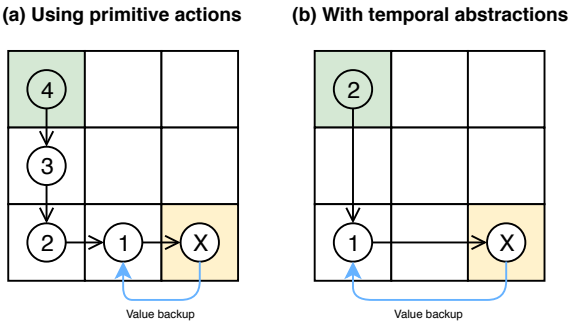
Anders and Andrew (2000) identified the benefit of utilizing different state abstractions for different sub-behaviors. The intuition behind this idea is that when performing different sub-behaviors, other aspects of the state space become relevant. Ignoring irrelevant features to the sub-behavior allows for the more efficient exploration, and learning of the different temporal abstractions.

3.3 HRL Advantages

Using both temporally extended actions and state abstractions, HRL is capable of providing significant improvements in various open RL challenges, such as credit assignment, exploration, and continual learning.

3.3.1 Credit Assignment

The issue of credit assignment is a long-standing challenge in RL. Credit assignment is tasked with figuring out which action had which impact on the overall perceived reward. Temporal and state abstractions can greatly increase the sample efficiency of RL by making credit assignment less challenging. Given a set of temporal abstractions, the agent is freed from the complexity of having to reason about which action to take on every individual step. The agent instead activates temporal abstractions which can run for multiple steps. When performing credit assignment by learning a value function, this has a profound effect on the efficiency of the value-backups (Sutton et al., 1999; McGovern et al., 1997). While utilizing primitive actions only, a value backup only goes back one step, thus when reaching a goal state, only the expected future value of the previously visited state



Figuur 3.2: **Value Backups**
(**a**) using primitive actions, and (**b**) using temporal abstractions. With temporal abstractions, an optimal policy can be found using fewer value backups.

gets updated. Using temporal abstractions, this reward gets propagated further back, allowing faster learning of a value function. This idea is demonstrated in Figure 3.2.

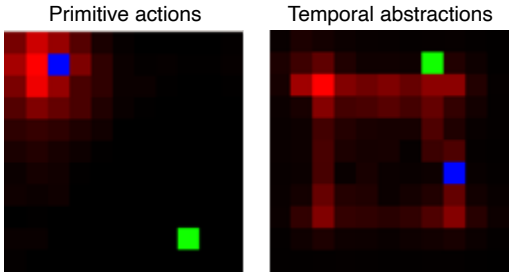
3.3.2 Structured Exploration

Improved sample efficiency through more structured credit assignment is capable of solving RL problems that previous flat RL algorithms could not solve. A recent ablation study (Nachum et al., 2019), empirically demonstrated that the most significant contribution of current HRL approaches lies in its capability to explore in a semantically meaningful and temporally extended fashion (Figure 3.3). Exploration, using only primitive actions often results in the over-exploration of states near the starting state of the agent. Using temporal abstractions for exploration allows the agent to explore the environment in a more structural way, which allows the agent to also explore more difficult-to-reach states. However, the opposite has also been observed. Given a suboptimal set of sub-behaviors, pathological exploration can easily worsen learning performance (Jong et al., 2008).

However, the opposite has also been observed. Given a suboptimal set of sub-behaviors, pathological exploration can easily worsen learning performance (Jong et al., 2008).

3.3.3 Continual Learning and Adaptation

Problems in RL are most of the time solved from scratch, starting without any prior knowledge. While finding a learning algorithm, capable of solving problems re-



Figuur 3.3: **Exploration through primitive actions vs through temporal abstractions** Exploration using only primitive actions, mostly explores states near the starting position of the agent (state visitation counts are represented using a red gradient). Temporal abstractions can help the agent to explore more distant states.

quiring long-term planning, without any priors, is certainly an interesting research direction. In the short term, the sample efficiency of RL can greatly be improved by building upon previously learned knowledge (Lake et al., 2017).

A policy learned without any abstractions is very difficult to transfer to a new problem (e.g., use a set of steering directions to navigate to a different location). In contrast, once learned, sub-behaviors can be transferred to solve different problems in similar environments (Tessler et al., 2017; Bacon et al., 2017; Heess et al., 2016; Schaul et al., 2015). Thus, the cost associated with learning sub-behaviors should not only be weighted on a single task but should possibly be considered over multiple tasks.

Tessler et al. (2017) for example presented a lifelong learning RL framework capable of learning different skills in the complex open-world game of Minecraft (Johnson et al., 2016). In a second stage, multiple skills are distilled into a single network in order to efficiently retain knowledge.

The ability of sub-behavior re-use is an important stepping stone towards agents which are capable of continual learning (Silver et al., 2013). More complex problems can be solved if an agent is efficiently able to improve upon what it already has learned from previous problems (Bengio et al., 2009).

3.4 HRL Challenges

Learning algorithms that utilize hierarchical compositions of temporally extended actions will have to define how these sub-behaviors should be discovered, how they can be developed, how efficient state abstractions can be learned, and how

they can be composed. In this section, we briefly describe these challenges.

3.4.1 Automatic Discovery of Abstractions

Early HRL approaches (Singh, 1992; Mahadevan and Connell, 1992; Maes and Brooks, 1990) utilized manually defined sub-behaviors. While they demonstrated performance increases by using sub-behaviors, the requirement of manually defining sub-behaviors heavily limited their scalability.

One of the most important questions of HRL consists of: how can we automatically discover meaningful sub-behaviors? A lot of empirical research (e.g., Eysenbach et al. (2019); Nachum et al. (2018b); Vezhnevets et al. (2017); Bacon et al. (2017)) has been conducted on algorithms that are capable of automatically learning meaningful sub-behaviors, from interaction with the environment, without any knowledge provided by an external expert, in a sample efficient way. Theoretically, it has been proven (Jinnai et al., 2019), that finding a small set of sub-behaviors in a limited number of steps is an NP-hard problem. Abel et al. (2020) studied which sets of state abstractions and temporal abstractions are capable of preserving the representation of near-optimal policies. However, this method requires full access to the underlying MDP.

In low-dimensional environments, a small discrete set of sub-behaviors can greatly improve the sample efficiency of the learning algorithm. However, in truly complex environments, a large continuous range of sub-behaviors will be required.

The most commonly used techniques to automatically discover temporal abstractions, either utilize special properties of different states (Machado et al., 2017; McGovern and Barto, 2001), or correlate trajectories with a sub-behavior random variable (Eysenbach et al., 2019; Haarnoja et al., 2018a; Nachum et al., 2018b; Vezhnevets et al., 2017). However, the definition of special states or the correlation between states and random variables often still relies on hand-crafted heuristics.

3.4.2 Policy Development of Abstractions

In order to use temporally extended actions, they need to receive enough suitable samples from the environment in order to become sufficiently developed. In the most promising HRL approaches, sub-behaviors emerge and develop policies simultaneously (Nachum et al., 2018b; Vezhnevets et al., 2017; Bacon et al., 2017). This however does not need to be the case. Sub-behaviors can also be discovered and trained using a staged approach (Tessler et al., 2017; Kulkarni et al., 2016a; McGovern and Barto, 2001). For example, the state space could be split into a few equal parts, with a sub-behavior assigned to each part. In a second phase,

the sub-behaviors can be developed by, for example, randomly activating them. However, as the number of sub-behaviors increases, more efficient development strategies will be required to make sure that all sub-behaviors are sufficiently trained. If multiple sub-behaviors are generalized by a single parameterized function, samples can be used to train multiple sub-behaviors at the same time. Similarly, off-policy methods can be used to simultaneously develop multiple abstractions (Sutton et al., 1999).

3.4.3 Decomposing Example Trajectories

Often, it is possible to obtain human demonstrations for different control tasks. HRL can maximize the utility of such demonstrations by decomposing them into sub-behaviors. Instead of learning a single task from demonstrations, the discovered sub-behaviors can be used as building blocks to solve a range of related tasks, potentially without the need for additional demonstrations.

In Imitation Learning (IL) (Peng et al., 2018; Ross et al., 2011; Argall et al., 2009; Bain and Sommut, 1999), the exploration problem of RL is made somewhat less complex, because the agent is equipped with a set of demonstration trajectories, which originate from a better than random policy (e.g., a human domain expert). While IL typically is used to solve a single problem, IL could be used to discover temporal abstractions, these abstractions might be re-used to solve multiple similar problems (Le et al., 2018; Fox et al., 2017; Krishnan et al., 2017).

Providing a reward function for a control problem is often a complex issue in itself (Ng et al., 1999). The idea of Inverse Reinforcement Learning (IRL) (Ho and Ermon, 2016; Ng and Russell, 2000) is that instead of manually specifying a reward function an agent should learn this reward function by observing demonstrations from an extrinsic agent. However, learning a single reward function for the entire problem is often too coarsely defined, or demonstrations might originate from a set of different reward functions, instead of one (e.g., when purely exploring multiple sub-behaviors). A single reward function might also be task- or environment-specific and does not allow for generalization over multiple problems. Hierarchical IRL aims to learn a composition of multiple smaller reward functions, which can in turn be used to train sub-behaviors using trial-and-error learning techniques (Pan et al., 2018; Krishnan et al., 2016).

3.4.4 Composition

Sub-behaviors alone are not enough to solve HRL problems. They need to be composed in order to form complex plans. A capable HRL algorithm will need to be able to select a favorable sub-behavior to use given a state.

There are two major approaches that are commonly used to learn compositions:

- **Bottom-up training**: the sub-behaviors are discovered and developed first. Once they have been sufficiently developed, they are frozen, and a composition policy is learned by sampling different sub-behaviors. This is the most straightforward way, as the higher level does not need to take into account that the outcome of the selected temporal extended actions might have changed. Sub-behaviors learned using a bottom-up approach, can often also be transferred to solve similar problems. However, in the first phase, time might be spent learning sub-behaviors that the higher level actually does not need.
- **Top-down training**: the higher level first selects a subgoal $g \in S$ it deems interesting in order to reach the overall goal. Once selected, the lower level is trained to reach the proposed subgoal. This approach is often more efficient, compared to training bottom-up in solving a single control problem. However, it needs to take non-stationary sub-behaviors into account. It is also often not straightforward to transfer sub-behaviors to different problems.

3.5 Problem-Specific Models

Initial research on HRL agents was focused on proving the benefits of using temporal abstractions in an online RL setting. In order to demonstrate the capabilities of HRL, highly problem-specific models were proposed. They offered intuitive, and often interpretable answers on how to model hierarchies of abstractions.

This problem-specific nature is often deemed to be difficult to learn automatically by a learning agent. Learning parts of problem-specific models often relies on the idea of information hiding and reward hiding (Dayan and Hinton, 1993). By either concealing parts of the observed state or reward, no sub-behavior has all the required information in order to solve the entire task. This forces different abstractions to focus on different parts of the task.

Problem-specific models however often also heavily rely on external knowledge provided by an expert, or task-specific structure present in the environment.

The most common problem-specific models are briefly discussed in the following subsections. A more in-depth survey on some of these frameworks and related early research has been previously conducted by Barto and Mahadevan (2003).

3.5.1 Feudal

Feudal Q-learning (Dayan and Hinton, 1993) is an approach, that makes use of a simple abstraction of the state space on multiple levels. Feudal Q-Learning has different managers assigned to different regions of the state space. This system is inspired by medieval society management: the king commands nobles, these nobles command knights, and so on. In Feudal Q-learning, a hierarchy of learning modules is constructed. At the top of this hierarchy is a manager, which is in charge of an abstracted state space, and sets out a task for a single lower-level worker within this space. The lower-level worker in turn is capable of taking action within this space. This lower-level worker is rewarded for successfully executing the received command, independently of the reward of the higher-level. Using reward-hiding, only the highest level manager uses the extrinsic reward signal to set out tasks for the level just below it. This approach has also been proven useful when confronted with large action spaces (Kumar et al., 2017).

While this is a highly interpretable approach, unfortunately, the Feudal model can only be utilized to solve a specific kind of problem in which the state space can be neatly divided (e.g., a floor plan).

3.5.2 Hierarchies of Abstract Machines

Parr and Russell (1998a) proposed an approach that composes the behavior of a HRL agent by utilizing different finite state machines, which are able to invoke each other. This approach is inspired by software development, in which functions call each other to manipulate the state of the program.

A machine in a Hierarchies of Abstract Machines (HAM) setting is defined by a transition function and a start function that is responsible for choosing the initial action of the machine. The different actions (machine states) each machine can take consists of: taking a primary action in the environment, calling another machine as a subroutine, or terminating the machine, and returning control to the machine that invoked it. The stochastic transition function is responsible for selecting actions depending on the environment state, and the previously executed action.

A HAM needs to be provided by an expert, it acts as a sketch for the solution, constraining exploration that needs to be done by the agent. The HAM-Q learning algorithm was proposed to transform the expert-provided HAM sketch into a policy capable of solving RL problems. This algorithm extends Q-learning (Watkins and Dayan, 1992), to not only consider the environment state in taking an action but also include the machine state. While a HAM is a highly interpretable and reusable model, hand-designing a HAM is often an infeasible task for complex problems.

3.5.3 MAXQ

The MAXQ-framework (Ghavamzadeh and Mahadevan, 2001; Dietterich, 2000) is a framework for representing decomposed value functions. A decomposition of the value function answers the question: what factors contribute to the overall expected cumulative future reward? The proposed MAXQ decomposition is hierarchical: solving the root task solves the entire control problem. MAXQ is able to model both temporal and spatial abstractions. A MAXQ decomposition consists of two different types of nodes:

- **Max-nodes** define different sub-behaviors in the decomposition and are context-independent
- **Q-nodes** are used to represent the different actions that are available for each sub-behavior and are specific to the task at hand.

The distinction between max and Q nodes allows max nodes to be shared by different tasks. For example, in the Taxi benchmark environment (Dietterich, 2000), the agent needs to pick up, and transport customers to their destinations, a *navigate* max node can be used by both a refuel and get-passenger sub-behavior.

Similar to the HAM-Q algorithm, a MAXQ-Q learning algorithm (Dietterich, 2000) has been proposed to learn policies for the different nodes.

A major difference between the HAM-model, and the MAXQ framework is its ability to handle stochastic environments. While the proposed method of executing a HAMQ-Q policy hierarchically (committing to sub-behaviors until termination), an alternative polling executing approach is proposed. In this alternative execution mode, each level of the hierarchy is evaluated at each time step. This allows the MAXQ-Q algorithm to operate more efficiently in highly stochastic environments.

The MAXQ-Q learning algorithm provides a way to learn how to use a decomposed value function. However, this approach is limited applicable, because the decomposition needs to be provided by an external expert.

Hengst (2002) proposed a method for automatically decomposing a value function within the MAXQ framework. Instead of relying on an expert to provide the decomposition, HEXQ is able to learn a hierarchy from scratch.

In the HEXQ algorithm, a different hierarchical level is considered for each dimension of the state space. The construction of the hierarchy starts by observing which dimension of the state space has the highest change frequency. In order to determine this, a random policy is used for an arbitrary amount of time steps. The experience gathered from this random policy is then used by the HEXQ algorithm to build a graph of state transitions. Extra attention is paid to transitions

not following a stationary distribution. States that exhibit such special transitions are called exit states. In these cases, the information provided by the current state dimension is not enough to make a decision and other parts of the hierarchy will need to decide how to handle these situations. States that are reached using these exit states are called entry states. In order to build usable state abstractions, the transition graph can be split into multiple regions. A region is defined, so each exit state should be reachable from each entry state assigned to the region. A different MDP can thus be considered for each region.

One limitation of the HEXQ approach is that it only considers a single state dimension on each level. The ordering heuristic might not sufficiently be capable of detecting what state features depend upon each other efficiently. While this approach works in a lot of simple domains, it might not find good solutions for more complex control problems with higher dimensional state spaces.

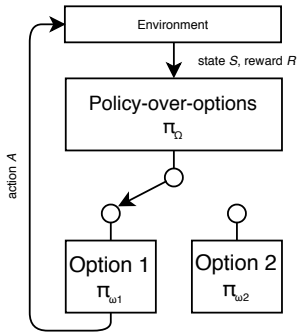
Another approach of automatically learning a MAXQ decomposition is called Variable Influence Structure Analysis (VISA) (Jonsson and Barto, 2006). VISA uses a Dynamic Bayesian Network (DBN), capable of modeling causal relationships between actions and state dimensions. Combinations of state dimensions and actions that cause important value function changes are considered sub-behaviors.

Hierarchy Induction via Models And Trajectories (HI-MAT) (Mehta et al., 2008) is also capable of discovering a MAXQ-decomposition. HI-MAT differs from VISA in that it also requires at least one single successful trajectory. Causal and temporal relationships among actions in this trajectory are analyzed in order to decompose the trajectory into multiple sub-behaviors. This leads to a decomposition that is more compact than those typically discovered using VISA.

3.6 Options

The problem-specific models presented in the previous section are difficult to automatically discover because of their non-generic nature, and complex architectures. The options framework (Sutton et al., 1999) provides an alternative framework to model temporally extended actions in a more generic way so that automatically learning sub-behaviors and their composition becomes more feasible in multiple settings, using the same learning algorithm. However, the ideas introduced by the reviewed problem-specific frameworks, also remain relevant in the options framework. This is due to the fact that most problem-specific frameworks can be represented as options.

In the options framework, the action space of the agent is extended with temporally extended actions called options. The SMDP formalism is used in order to model control problems that utilize options. An option is considered to be semi-Markov if



Figuur 3.4: **The Options Framework**
Based on the current state a policy-over-options is consulted in order to decide which option to activate. The policy of this option will then be used to sample actions until its termination condition is triggered.

its policy does not only depend on the current state of the MDP but also depends on the set of observed states and rewards since the option was invoked. This set could for example be used to terminate an option if it failed to satisfy the termination condition within a number of steps.

Options represent closed-loop sub-behaviors (Figure 3.4), which are carried out for multiple timesteps until they trigger their termination condition. Options are considered closed-loop systems because they adapt their behavior based on the current state. This is in contrast to open-loop systems, which do not adapt their behavior once initialized when confronted with a new state. It is often more realistic to model sub-behaviors as closed-loop systems than using open-loop sub-behaviors. For example, while driving a car, if we would be committed to an open-loop option, we would not deviate if we encounter danger, a closed-loop option will alter its action based on the current state.

Using a well-defined set of options will require the agent to make fewer decisions when solving problems (Mann and Mannor, 2014; Silver and Ciosek, 2012; Precup and Sutton, 1997). The usage of options in an RL setting has been shown to speed up learning. For example, (Tessler et al., 2017; Brunskill and Li, 2014) demonstrated options as a way to summarize knowledge as an essential building block in a lifelong-learning setting. Guo et al. (2017) demonstrated the increased performance of using temporally extended actions using importance sampling.

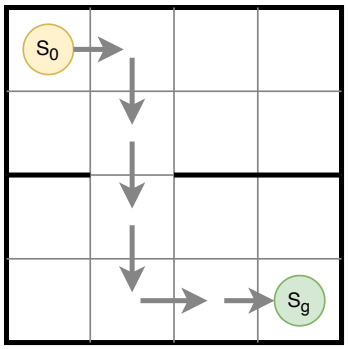
Various algorithms make use of the options framework, in the remainder of this section we first discuss the different components of the framework. Additionally, we review various algorithms capable of automatically discovering options by interacting with the environment, and how a policy-over-options can be found in order to compose options appropriately.

3.6.1 Framework

An option can be defined as a tuple $\omega = (\mathcal{I}, \pi, \beta)$:

- $\mathcal{I} \subseteq \mathcal{A}$ is the initiation set, containing all states in which the option can be initiated.
- $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$ is the intra-option policy, determining the action-selection of the option based on the current state (and optionally the set of states, since the option was invoked).
- $\beta : \mathcal{S} \rightarrow [0, 1]$ makes up the termination condition, which determines when the option will halt its execution.

In Figure 3.5 an example option is represented. In this example, the initiation set and termination condition are represented as single states, and the intra-option policy is represented by the arrows. This type of option with a single initiation- and termination state is often called a point option.



Figuur 3.5: **An example option policy**
The option policy takes the agent from the initiation state S_0 located in one room, to the termination state S_g in another room.

3.6.1.1 Initiation Set

The initiation set of an option defines the states in which the option can be initiated. Different approaches are typically used to define the initiation set. A commonly used approach defines the initiation set as all states from which the agent is able to reliably reach the option its termination condition when following the intra-option policy, within a limited amount of steps. It is also usual to assume that for all states in which the policy of the option can continue, it can also be initiated.

For example, (Konidaris and Barto, 2009b; McGovern and Barto, 2001) uses a logistic regression classifier to build an initiation set. States that were observed up to 250 time steps before triggering the option termination were labeled as positive initiation states for the selected option, states further away in the trajectory were used as negative examples.

An alternative approach consists of defining the initiation set of an option as the complete state space. Instead of using the initiation set in order to determine which option to activate, a policy-over-options (Sutton et al., 1999) is used to determine which option to initiate.

In a continuous state space setting, it does not make sense to use individual states as the initiation set. Reaching a single specific state in a continuous state space is very unlikely, so in this case, the initiation set can be defined as a region (Neumann et al., 2009; Konidaris and Barto, 2009b).

3.6.1.2 Termination Condition

The termination condition decides when an option will halt its execution. Similarly to the initiation set, a set of termination states is often used. Reaching a state in this set will cause the option to stop running. Termination states are better known as subgoal states. Finding good termination states is often a matter of finding states with special properties (e.g., a well-connected state or states often visited on highly rewarded trajectories).

The most common termination condition is to end an option when it has reached a subgoal state. However, this can lead to all kinds of problems. The option could for example run forever when it is not able to reach the subgoal-state. To solve this, a limitation of the allowed number of steps taken by the option policy is often also added to the termination condition.

A termination condition has also been considered when the agent is no longer active in its initiation set (Şimşek and Barto, 2004). In addition, gradient-based approaches have been proposed, capable of maximizing long-term return, given a set of options (Comanici and Precup, 2010) or while simultaneously also learning the option policies (Bacon et al., 2017).

Similar to the initiation set, if the state space is continuous, the termination condition should be defined as a function, or as a region inside the state space. This region will decide when the option policy is leaving the state space it was assigned to by the initiation set.

3.6.1.3 Intra-Option Policy

If the initiation- and termination-set are specified, the internal policy of the option can be learned by using any RL method. The intra-option policy can be seen as a control policy over a region of the state space.

An important question that needs to be addressed when learning intra-option policies from experience, is how these policies should be rewarded. The extrinsic reward signal is often not suitable in this case, because the overall goal does not necessarily align with the termination condition of the option. Alternatively, the intra-option policy could solely be rewarded when triggering its termination condition. However, various denser intrinsic reward signals could also be used. For example, if the termination condition is based upon a special characteristic of the environment, this property might serve as an intrinsic reward signal.

Intra-option policy learning can both happen on-policy and off-policy. With on-policy learning, only the policy of the invoked option is updated. Sutton et al. (1998) explores off-policy option learning methods that are able to improve the policy of an option, even if it is currently not active.

3.6.1.4 Policy-over-Options

A policy-over-options $\pi(\omega|s_t)$ selects an option $\omega \in \Omega$ given a state s_t. This additional policy can be useful to select the best option when the current state belongs to multiple option initiation sets. It can also be used as an alternative to defining an initiation set for each option.

The most often used execution model is the call-and-return model. This approach is also often called hierarchical execution. In this model, a policy-over-options selects an option according to the current state. The agent follows this option until the agent triggers the termination condition of the active option. After termination, the agent samples a new option to follow.

An alternative model called the one-step-options model, or also sometimes called non-hierarchical execution model, queries the policy-over-options on every single timestep, allowing switching options, even if the option is not fully terminated yet. For example, Mankowitz et al. (2014) suggests switching options when the expected total future value of an option other than the current executing option has become higher. However, options should mostly be able to run for a certain amount of steps in order to be useful. Harb et al. (2018) incorporated a termination deliberation cost in order to prevent options switching on each time step.

When considering a policy-over-options, we can identify different forms of optimality:

- **Hierarchical-optimal**: a policy that achieves the maximum highest cumulative reward on the entire task.
- **Recursive-optimal** (Dietterich, 2000): the different sub-behaviors of the agent are optimal individually.

A policy that is recursive-optimal might not be hierarchical-optimal. It is possible that there exists a better hierarchical policy, where the policy of a sub-task, might be locally suboptimal, in order for the overall policy to be optimal. For example, if a sub-task consists of navigating to the exit of a room, the policy is recursive-optimal when the agent only fixates on this sub-task. However, a hierarchical-optimal solution might also take a slight detour to pick up a key or charge its battery. These diversions negatively impact the performance of the sub-task, but improve the performance of the overall task.

3.6.2 Option Subgoal Discovery

The most common approach of automatically discovering options is focused on finding good termination conditions consisting of reaching a single state. These subgoal states often exhibit special characteristics. For example, a doorway of an elevator is a special state because it provides access to otherwise impossible-to-reach areas. This approach also significantly helps efficient exploration. Easy access to these important states, facilitated by the intra-option policy, will allow the agent to explore further.

What makes a state a good subgoal? This is a difficult question to answer as there are a lot of often conflicting interesting properties of states that can be used to identify subgoals states. In the following section, we review some properties that have successfully been used to identify useful subgoal states from collected experience in the environment.

3.6.2.1 Landmark States

A landmark state is a cognitive reference point. It is common for people to organize spatial information hierarchically using such landmarks. Landmarks used by humans, in order to come up with complex plans, are often stored in a low-dimensional representation (e.g., a rough outline of the shape of the Eiffel-tower, instead of a detailed picture).

The Hierarchical Distance to Goal (HDG) algorithm (Kaelbling, 1993) is capable of navigating a complex environment by navigating between landmarks. These landmarks are cluster centers of regions. These regions need to be specified upfront. The agent will first transition between landmarks to navigate to the region which also contains the goal. Once successfully transitioned to the goal region, primitive actions will be used to navigate to the goal state. In order to efficiently navigate between landmarks, or within a single region, the amount of steps required to navigate from one state to another, the distance to goal is estimated.

Landmarks provide an interesting way to navigate between vastly different areas of the state space. However, because these areas need to be provided by an expert, this approach does not scale well to large state spaces and does not allow the transfer of sub-behaviors to different environments.

3.6.2.2 Reinforcement Learning Signal

Digney (1998) used the reinforcement learning signal in order to identify useful subgoal states. States with a high reinforcement signal gradient are non-typical states and are considered useful states used in more complex navigation tasks. This approach is however limited because it is only applicable in environments with a dense enough reward signal.

3.6.2.3 Bottleneck States

Besides the reinforcement learning signal, Digney (1998) also considered using a history of the visitation frequencies in order to discover subgoal-states. Bottleneck states are states that are frequently visited on successful trajectories, but not on unsuccessful trajectories.

An example of a bottleneck state could consist of a state where the agent picks up a key or utilizes a door. These states are essential in trajectories that reached the overall goal, while trajectories of failed attempts might not contain these states. Bottlenecks discovered near the initial position of the agent are especially interesting, as they greatly benefit the exploration of areas further away from the initial position.

Automatically discovering bottleneck states can be done by keeping track of the visitation counts of states in successful trajectories. However, when using this approach, states near the starting position of the agent will be more often selected as potential bottleneck states, because more exploration is often done near the starting position of the agent. To avoid this bias, McGovern and Barto (2001) suggested only counting the first visits of states over a set of successful trajectories.

Trajectories are often considered in multiple instances of the same environment with different goals. For example, Stolle and Precup (2002) proposed instantiating multiple instances of the same environment with different goal states. States visited on successful trajectories across instances are considered to be bottleneck states.

McGovern and Barto (2001) described the problem of discovering bottleneck states, as an application of multiple-instance learning (Dietterich et al., 1997). Individual trajectories are considered bags, when a trajectory is successful it is considered a positive bag, when it is unsuccessful it is considered a negative bag. Diverse density (Maron and Lozano-Pérez, 1998) was used to discover individual subgoal states.

Kulkarni et al. (2016b) introduced a method also capable of discovering bottleneck states in high-dimensional state spaces. The introduced algorithm used a learned approximate successor map. Such a map represents a state in terms of its expected future state occupancy, called the successor representation (Dayan, 1993). When sufficiently developed by following a random policy, a large set of samples from the successor representation can be used to discover bottleneck states.

Bottleneck states are an interesting way of discovering subgoal states because they provide easy access to key states in the environment. However, this approach cannot be utilized in all environments. Bottleneck states often correspond with doors, hallways, or elevators. However, some environments naturally lack bottleneck states (e.g., joint positions of a robot arm).

3.6.2.4 Access States

Access states allow the agent to transition to regions of the state space that are otherwise difficult to reach. Example access states include: a doorway between two rooms or an elevator. Access states are natural in navigation tasks, but can also be found in other state spaces: for example, picking up a screwdriver will unlock all kinds of attaching possibilities. Access states are similar to bottleneck states but do not require successful trajectories, which are often difficult to collect. Instead of relying on the reward signal (bottleneck states), access states rely on a measurement of novelty.

Relative novelty (Şimşek and Barto, 2004) can be used to identify access states. Relative novelty considers the novelty of the predecessor states, and the successor states. Subgoal candidates have a different novelty score than regular states. For a regular state, the novelty of neighboring states will be more or less the same. However, for a difficult-to-reach door or elevator state, the novelty of states that can be reached from this state will be very different. Relative novelty (Şimşek and Barto, 2004) can be used to identify access states. Relative novelty considers

the novelty of the predecessor states, and the successor states. Subgoal candidates have a different novelty score than regular states. For a regular state, the novelty of neighboring states will be more or less the same. However, for a difficult-to-reach door or elevator state, the novelty of states that can be reached from this state will be very different.

Goel and Huber (2003) proposed a similar approach where funnel states are identified, these states have a significantly larger number of predecessor states that lead to them, while they only have a limited number of known successor states.

3.6.2.5 Graph Partitioning

The MDP model, represents the RL problem as a graph. Techniques used to partition graphs in general, have also been utilized to discover options.

The Q-Cut algorithm (Menache et al., 2002), models trajectories utilized by an agent in a graph structure. The nodes in this graph represent the different states, edges are concerned with modeling state transitions. A min-cut approach (Waissi, 1994), will try to discover a set of edges that if we would remove them, the graph would be split into two unconnected graphs. This procedure can be applied iteratively, resulting in multiple detached graphs. Detecting such edges in the learned state-transition graph structure can lead to the discovery of bottleneck states.

Şimşek et al. (2005) introduced a similar approach called L-Cut. This method partitions local state transition graphs, in order to discover access states that can be utilized as useful subgoal states. The difference with Q-Cut is that L-Cut does not rely on the entire transition-graph, but utilizes a local view of the graph, making it less computationally demanding, and better scalable to larger state spaces.

Machado et al. (2017) demonstrated that a learned representation with Proto-Value Function (PVF) (Mahadevan, 2005) can be used in order to discover options. By utilizing the transition matrix of the underlying MDP, PVFs can be obtained. A PVF tries to capture the topology of the state space, facilitating the structural decomposition of large state spaces. The options found in the eigen-options framework (Machado et al., 2017) each can be seen as traversing one of the dimensions found in the learned representation. The intrinsic reward linked to traversing such a dimension is defined as the eigenpurpose of the option. The intra-option policy which is derived when following the eigenpurpose is called the eigenbehavior.

Machado et al. (2018) extended the eigen-option framework to also be applicable when a linear representation is not available by using a successor representation (Dayan, 1993), to estimate a topology of the state-space. This extension also allows the discovery of eigen-options in stochastic environments and allows discovery without the necessity of a handcrafted feature representation. The successor

representation can be approximated using deep neural networks (Kulkarni et al., 2016b), which allows eigen-options to be discovered in high-dimensional state spaces.

3.6.2.6 State Clustering

Similar states can also be grouped using clustering techniques. States that facilitate navigating between different clusters are natural bottlenecks. Lakshminarayanan et al. (2016) proposed using a spectral clustering algorithm PCCA+, that is capable of simultaneously partitioning the state space, and returning connectivity info between different partitions from sample trajectories.

3.6.2.7 Skill Chaining

Previously described methods for automatically discovering subgoals are often limited to operating only in discrete state spaces. In a continuous space, single states are often never visited multiple times. Konidaris and Barto (2009b) proposed an algorithm capable of discovering option-based sub-behaviors in a continuous state space. Instead of utilizing termination states in the options framework, skill chaining defines termination regions for the different options. Similarly, the initiation condition is also defined as a region.

Given a termination region, the initiation region of the options can be considered a classification problem. Given a set of sample trajectories following a learned flat policy, states that are capable of reaching the termination region within a limited amount of steps are positively classified.

The first option will have the environment goal as its termination region. Once the initiation set of this option has been learned, a second option can be learned. The termination region of this option will be the initiation region of the previously learned option. This procedure is repeated until a chain of options is discovered up to the agent's starting position. A more complex skill tree could be learned similarly, allowing the discovery of multiple solution paths.

However, in order to build a skill chain or tree, a policy first needs to be trained which is capable of reaching the end goal, in order to generate meaningful trajectories. Because of the requirement of such a policy, the usefulness of skill chaining remains limited to the transfer learning case.

3.6.3 Motion Templates

Motion templates (Neumann et al., 2009) are options that can be parameterized in order to adapt the behavior of its intra-option policy. This is often useful in a continuous state space. A motion template could for example be discovered for throwing a ball. The exhibited force and angle might be parameters of this template. Learning a single policy for each possible combination of force-angle would be infeasible. Using motion templates allows the generalization of sub-behaviors. da Silva et al. (2012) proposed a method for learning motion templates from experience using classifiers, and non-linear regression models.

3.6.4 Macro-Actions

Another approach for discovering temporally extended actions consists of trying to discover interesting sequences of actions, called macro-actions. This approach differs from the options framework, in that macro-actions are often open-loop. The intra-option policy most commonly consists of a fixed set of actions and does not depend on the current state.

The STRategic Attentive Writer (STRAW) architecture (Vezhnevets et al., 2016) is capable of discovering macro-actions as commonly occurring sequences of actions (multi-step plans) directly from the extrinsic reward signal. Once activated, STRAW follows the macro-action for a variable number of steps, without updating it. Instead of the traditional policy, which selects actions one at a time, STRAW selects sequences of actions and learns when to shift course. The problem thus becomes finding out when decisions need to be made and finding macro-actions that an agent can follow between decision points. Attentive writing (Gregor et al., 2015) is used to determine what part of a plan is relevant to determine further sequences of actions. The differentiable of this algorithm makes it possible to learn when to commit to the current action plan or when to re-evaluate.

An adapted architecture called STRAWe (Vezhnevets et al., 2016) was proposed with added noise, encouraging exploration.

Macro-actions discovered by STRAW, in a set of Atari benchmark games (Bellemare et al., 2013), corresponded to interpretable sub-behaviors such as avoiding enemies, and navigating between game elements. The commitment plan efficiently showed a preference for shorter macro-actions when faced with fast-paced games, or when agility is required (e.g., when directly facing an enemy).

Fine Grained Action Repetition (FiGAR) (Sharma et al., 2017) is similar to STRAW in that it selects multiple actions based on a single observation. FiGAR however decides on an action and the number of times it should be repeated. FiGAR works

as an extension to another RL algorithm. While showing that this method can improve the performance of an already well-performing agent, a limitation is its inability to respond to sudden changes while committed to repeating an action.

3.6.5 Using Options in High-Dimensional State-Spaces

Research on option discovery has mostly been focused on low-dimensional state spaces. However, the options framework has also been demonstrated to be capable of learning options when using function approximation in high-dimensional state spaces.

Kulkarni et al. (2016a) studied the construction of option-based hierarchical agents, given a set of expert-provided termination conditions, in the form of the pixels of subgoal states. The resulting Hierarchical-DQN (H-DQN) algorithm uses a two-layered approach, in which the low-level controller uses DQN (Mnih et al., 2015) in order to learn a different intra-option policy for each of the provided termination states. A pre-training phase is used first in which sub-behaviors are randomly activated. This will allow the options with easier-to-reach termination conditions to become sufficiently developed.

During a second training phase, a higher-level controller learns a composition of the different sub-behaviors, also using the DQN algorithm, while also jointly further training the individual sub-behavior policies. Because of the pre-training phase, the agent is now capable of exploring harder-to-reach subgoal states. This two-layered approach was able to achieve progress, on hard exploration navigation tasks, in high-dimensional state spaces, in which previously no progress had been made.

Tessler et al. (2017) proposed a similar architecture called Hierarchical Deep Reinforcement Learning Network (H-DRLN), which utilizes a form of curriculum learning (Bengio et al., 2009). The agent first learns to solve simpler sub-tasks, and successfully re-uses this knowledge as sub-behaviors in more complex tasks. This was demonstrated in the game Minecraft. Different Deep Skill Networks (DSN) were trained to solve different sub-problems, such as navigation tasks, or object pick-up tasks. In a second phase, the H-DRLN agent can solve combinations of slight variations of the sub-problems, in a more sample-efficient way than DQN (Mnih et al., 2015). Additionally, a form of policy distillation (Rusu et al., 2015) is proposed in order to merge multiple skills into a single network. This Distilled Multi-Skill Network requires fewer computing resources than the individual skill networks.

While this approach is limited because of its heavy dependency on an expert who needs to design individual problems to train the sub-behaviors, this approach demonstrates the capability of a hierarchical agent to be capable of transfer-

ring knowledge between tasks, paving the way for a lifelong learning framework (Silver et al., 2013).

3.6.6 Option Discovery as Optimization Problem

Previous discussed approaches separated the issue of discovering options and learning a policy-over-options. This approach of bottom-up learning risks wasting time learning sub-behaviors that might not be required in order to solve the problem at hand. Formulating option development and discovery as part of an optimization problem tasked with optimizing total future reward is an alternative approach that allows options and a policy-over-options to be learned end-to-end.

The Hierarchical Relative Entropy Policy Search (HiREPs) algorithm (Daniel et al., 2012) extends the Relative Entropy Policy Search (REPS) algorithm (Peters et al., 2010) to the hierarchical setting. The REPS algorithm addresses the problem of maximizing the expected reward of a policy while bounding the information loss (relative entropy) due to policy updates.

HiREPs uses the same information theoretic regularizer as REPS, but also includes learning options as a latent variable estimation problem. HiREPs learns options that are separable in the action space, minimizing overlap. This is achieved by estimating the probabilities that actions have been sampled by the different options, and updating the weight according to these probabilities. This should lead to options that generate different actions in similar states.

Daniel et al. (2016) proposed a framework capable of inferring option components from sampled data using expectation maximization. All components of the options are represented as distributions. HiREPs was utilized in this framework to sample data.

The Option-Critic (OC) algorithm (Bacon et al., 2017) is an end-to-end framework capable of jointly discovering and developing options without using prior knowledge. This approach was inspired by the Actor-Critic framework (Sutton, 1984). The actor part in the OC framework consists of multiple intra-option policies. The critic part is capable of assessing the discounted future value of options and actions. An option is selected by a policy-over-options, and runs until the agent triggers the stochastic option termination condition. The gradient of the termination conditions uses the advantage (Baird, 1993) regarding the future value of the option, compared to other options. Options that exhibit a high advantage over other options are updated using this gradient to run longer.

Harb et al. (2018) proposed to add a deliberation cost to this gradient-update, in order to avoid options collapsing into single-step options. This deliberation can be interpreted as how much better an option needs to be in order to switch. Unfortu-

nately, this deliberation parameter needs careful tuning.

Klissarov et al. (2017) introduced the Proximal Policy Option-Critic (PPOC) architecture. Extending Option-Critic to become applicable on continuous tasks, by incorporating the PPO algorithm (Schulman et al., 2017). PPOC uses a stochastic policy-over-options.

Harutyunyan et al. (2019) proposed the Actor-Critic Termination Critic (ACTC) which, similarly to Option-Critic, focuses on the termination part of the options. By concentrating the termination probabilities of options around a small set of states, higher-quality options in terms of learning performance, and intuitive meaning, can be discovered.

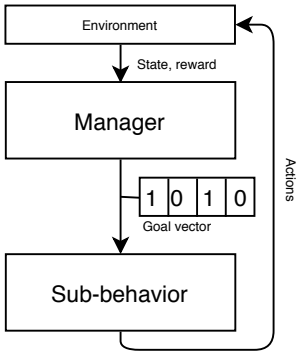
The Option-Critic algorithm is capable of learning options end-to-end, even in high-dimensional state spaces, without any expert knowledge, or additional intrinsic reward structures. Option-Critic has shown similar results to DQN (Mnih et al., 2015) in the Atari benchmark (Bellemare et al., 2013). While this is an important stepping stone in further advancing the applicability of the options framework, additional research is required in order to further stabilize automatic end-to-end option learning.

3.6.7 Options as a Tool for Meta-Learning

In a meta-learning approach (Wang et al., 2017a; Finn et al., 2017; Duan et al., 2016b), we search for adaptability. An agent is not trained to solve a single problem, but rather optimized to quickly solve unseen, similar problems. This method is often presented as *learning to learn*. Options learned, using meta-learning techniques, allow options to become less focused on a single problem, and should facilitate the transfer of options to novel problems.

For example, the Meta Learning Shared Hierarchies (MLSH) (Frans et al., 2018) algorithm, learns a set of sub-behaviors, by utilizing a distribution of different tasks. For each specific task, a policy-over-options is learned. Individual options are optimized in order to learn a good policy-over-options on a new task as quickly as possible. MLSH uses an incremental approach, in which a single task is sampled first. In this initial stage, the existing options are challenged as-is to solve this task. In the second stage, both the intra-option policies and the policy-over-options are updated jointly. Afterward, a new task is sampled, and the procedure is repeated until convergence.

HiPPO (Li et al., 2020), similarly focuses on how sub-behaviors can be made more robust to changes in the environment by utilizing a random runtime of the different options. HiPPO does not require a complex training scheme and a single task can be used as the base for a different task.



Figuur 3.6: **The Goal-conditional Framework**

Sohn et al. (2020) proposed a method capable of learning an adaptation policy by inferring the underlying subtask graph from a limited number of sample trajectories.

3.7 Goal-Conditional

It has been deemed difficult to propose learning methods for the presented problem-specific frameworks. The options framework provided a powerful generic language that spurred the development of various learning algorithms. However, the biggest disadvantage of the options framework is that it is difficult to scale to support numerous sub-behaviors. In addition, training options is often inefficient, because most commonly only one option is trained at the time, no components are shared between options, and options are often developed independently of the upstream tasks.

The goal-conditional framework models sub-behaviors differently. In order to support a large number of sub-behaviors a goal-vector $z \in Z$ is utilized to express different sub-behaviors. This goal vector is utilized to communicate a sub-behavior, activated by a higher-level manager, to a lower-level worker. This idea is illustrated in Figure 3.6. This goal vector can be discrete in order to express a limited number of abstractions, but it is also possible to use a continuous vector to express an infinite number of possible abstractions.

In this framework, experience collected pursuing one goal will also be useful for other related subgoals. This is due to the fact that goal-conditional algorithms often share common components such as a state representation. This makes goal-conditional algorithms capable of generalizing sub-behaviors, even for previously

unseen states and untested sub-behaviors. Expressing sub-behaviors using a goal vector, makes it possible to simultaneously learn a large number of sub-behaviors. Once the temporal abstractions are sufficiently developed, solving RL control problems in a goal-conditional context becomes a matter of sequencing appropriate goal vectors. This new problem ($\pi(s) : S \rightarrow Z$) should be considerably less difficult than solving the original problem ($\pi(s) : S \rightarrow A$).

When designing goal-conditional algorithms, two important questions need to be addressed. The first question relates to the way goals are represented. Multiple ways of representing goals have been proposed ranging from utilizing the full state space (Nachum et al., 2018b) to smaller latent spaces (Haarnoja et al., 2018a; Vezhnevets et al., 2017). When the dimensionality of the goal space Z becomes too large, it will be difficult to train sub-behaviors capable of expressing all possible sub-behaviors, however, when utilizing a smaller goal space than the original state space Z, some behaviors are, possibly not expressible with the chosen goal-space.

Once a decision has been made on how to represent sub-behaviors, a second major design decision needs to be made on how to sample various sub-behaviors in order to facilitate efficient training.

Similar to how options provided a more generic template to model the reviewed problem-specific frameworks, the goal-conditional framework can also be used to model options. Thus, ideas on discovering and developing abstractions, introduced in the previous frameworks, are also relevant in the goal-conditional framework.

In the following subsections, we provide an overview of different proposed goal-conditional techniques which decompose the state or the reward function or rely on unsupervised entropy to discover abstractions.

3.7.1 General Value Functions

In the classic RL approach a value function $V_\pi(s_t)$ is often learned to determine the total future expected reward starting in state s_t, while following policy π. The idea of a General Value Function (GVF) (Sutton et al., 2011) is to apply the same learning ideas used to learn a single value function, to a discrete multitude of different prediction targets, besides the extrinsic reward signal. Examples of such targets are, learning how many steps there are before an episodic control problem terminates, or learning the distance before hitting a wall. Learning different value functions can be seen as a way to build general knowledge about how different aspects of the environment can be manipulated. The intuition behind this idea is that if we know how our environment works, we should be able to easier achieve goals in this environment. Different value functions can essentially be utilized as abstractions, allowing the agent to reason on a higher level of abstraction.

In order to develop these different value functions, off-policy learning is often used so that an agent can learn multiple targets simultaneously using the same experience while not necessarily maximizing for all different targets.

The Horde algorithm (Sutton et al., 2011) is an example, of an algorithm capable of learning different targets. Horde utilizes a large number of independent sub-agents called demons. Each demon is responsible for learning a single predictive target about the world. Each demon has a policy, reward function, termination function and termination reward function. For the termination function and termination reward, termination refers to an interruption in the normal flow. A termination condition for a specific hydration-management value function could for example be: *ran out of water*.

Similar to Horde, Bengio et al. (2017) uses an autoencoder (Hinton, 2006; Rumelhart et al., 1985) to disentangle various factors of variation in the state space. For each discovered factor of variation, a different policy is learned that maximizes change on a single factor of variation.

The Horde architecture should be seen as a non-trivial exercise in knowledge representation and scalable abstraction learning. However, it only hints at how the discovered abstractions can be utilized together in order to solve entire RL control problems.

The Universal Value Function Approximation (UVFA) architecture learns a single value function $V(s, z)$ where the goal-state z is a parameter. Because learning a different value function for each state would be infeasible in a high-dimensional state space, UVFA uses factorization techniques and function approximation in order to develop $V(s, z)$. This architecture can be trained using supervised learning, by factoring observed values into an embedding for the state, and a separate embedding for the goal state. UVFA has also been demonstrated to work in an RL setting, by using Horde (Sutton et al., 2011). UVFA allows Horde to generalize to unseen goal-state predictive targets.

While the UVFA framework is capable of efficiently generalizing navigation to different subgoal states, it does not provide a way of addressing the issue of which subgoal should be targeted given the current state, or which subgoal states should be used to train the Horde, these still need to be provided by an expert. Levy et al. (2019) developed a learning algorithm that is capable of automatically selecting subgoal states, by utilizing a multi-level architecture in which each level (besides the lowest level) outputs subgoal states. By limiting the number of steps the agent has to reach a subgoal, the agent learns which subgoal states are reachable from the current state.

Hybrid Reward Architecture (HRA) (Van Seijen et al., 2017) takes as input a decomposed reward function similar to the MAXQ algorithm (Dietterich, 2000) and learns a separate value function for each of these functions. Because each part of

the reward function only depends on a subset of all features, approximating these individual value functions becomes more feasible.

A separate agent (and corresponding value function) is assigned to each separate reward function, similar to the Horde architecture. Each agent can have its own Q-function, or one Q-function with multiple heads can be learned, so that a shared state representation can be utilized. The output of the individual value functions is combined to estimate a single value for each state/action pair, which can be used to solve the original control problem. However, during training, the agent does use the individual outputs of the different reward functions.

The original research leading up to the UVFA architecture focused on the development of a goal-conditional framework to guide further research. Next to the question of how multiple predictive targets can be learned from a stream of experience, also different possible types of auxiliary targets have been studied. For example, Jaderberg et al. (2017) introduced two types of unsupervised auxiliary predictive targets: pixel-control and feature-control.

Using pixel control, the agent learns a separate policy that is capable of maximizing observed pixel change in cells of a non-overlapping grid that is placed over the original state observation. Such a cell in a grid could for example contain a door, by opening it, the agent will drastically change the pixels of this cell.

Similarly, feature control learns a separate policy that aims to activate neurons in the network of the behavioral policy. The intuition behind this idea is that, when sufficiently developed, these neurons should represent some useful features that impact the policy of the agent. Thus, if the agent learns to manipulate these neurons, it should be able to change the environment in meaningful ways.

The UNsupervised REinforcement and Auxiliary Learning (UNREAL) architecture (Jaderberg et al., 2017) uses pixel-control and feature-control as auxiliary rewards which can be optimized together with an A3C agent (Mnih et al., 2016).

Auxiliary rewards have also been used to drive exploration. For example, the Scheduled Auxiliary Control (SAC-X) architecture (Riedmiller et al., 2018) uses auxiliary tasks such as minimizing/maximizing the distance between objects, maximizing velocity, or activating a touch-sensor in order to actively drive scheduled exploration. They demonstrated that complex robotic manipulation behavior can be learned from sparse extrinsic reward signals.

Modeling multiple value functions capable of effectively representing different temporal and state abstractions proves to be an efficient way to integrate expert knowledge into an RL agent. This method effectively allows the agent to reason on a higher level of abstraction. However, the often, very domain-specific required expert knowledge, required to build the architectures, is also the biggest limitation of this approach.

3.7.2 Information Hiding

Information hiding (Dayan and Hinton, 1993) was a popular early approach for training problem-specific frameworks. However, more recently information hiding has also been used to facilitate training goal-conditional agents. As no single part of the architecture has access to all available information, different parts need to collaborate. While the GVF framework focuses on decomposing the reward function, information hiding focuses on decomposing the state space.

Heess et al. (2016) proposed an algorithm in which the lower-level components only have access to task-independent proprioceptive information (e.g., joint angles, velocities, or haptic information), the higher-level component has access to all available information, including exteroceptive observations (e.g., vision and audio). This approach draws inspiration from biology in which the brain typically composes plans utilizing exteroceptive observations. While the lower-level systems like for example the spinal cord, control task-independent sub-behaviors, which only utilize task-independent proprioceptive information.

In this architecture the high-level policy is recurrent, and the low-level policy utilizes a feedforward network. The high-level module utilizes a modulator control signal which is updated every k-frames. This signal is used to activate different sub-behaviors. A pre-training phase is utilized in which the low-level controller acquires generic locomotion sub-behaviors first, using a shaped reward signal (e.g., move in all available directions). Once sub-behaviors have been developed they are frozen, and a higher-level policy can be trained to modulate sub-behaviors in order to maximize extrinsic reward.

Using information hiding is similar to providing a decomposed reward function, an interesting way of incorporating expert knowledge. However, decomposing the state space has often the additional benefit of being task-agnostic, which should facilitate the transfer of the learned abstractions.

3.7.3 Unsupervised Learning in HRL

In various RL sequential decision problems, feedback received from the environment through the reward signal is often very sparse, and denser reward signals are difficult to construct (Ng et al., 1999). Unsupervised learning is a subfield of machine learning, concerned with discovering patterns without any feedback signal. A typical example is clustering data elements in coherent groups.

Unsupervised learning methods have also been proven useful in an RL context. Unsupervised learning allows the agent to learn sub-behaviors without utilizing information hiding or the reward signal. For example, a popular unsupervised

method is to maximize entropy $\mathcal{H}(\pi(\cdot|\mathbf{s}_t))$ as an intrinsic reward in addition to the extrinsic reward $r(s_t, a_t)$ (Ziebart et al., 2008; Todorov, 2007):

$$J(\pi) = E_{\tau \sim \rho_\pi(\tau)} \left[\sum_t r\left(s_t, a_t\right) + \alpha \mathcal{H}(\pi(\cdot|s_t)) \right] \tag{3.1}$$

RL solutions that only optimize for cumulative future expected reward, often risk finding a policy that is only locally optimal. By optimizing an agent for future value, while also being as random as possible, the agent becomes less likely to be stuck in a local optimum, more robust to permutations, and often manages to explore more efficiently (Haarnoja et al., 2017).

In HRL, acting as random as possible is utilized to find a diverse set of sub-behaviors. Because this method often results in a large amount of discovered sub-behaviors, the goal-conditional framework is an ideal candidate to model this kind of sub-behaviors.

Besides entropy, other forms of unsupervised learning can be utilized to discover sub-behaviors. For example, Nachum et al. (2018a), considered how well a learned goal representation is capable of expressing a near-optimal policy as an additional optimization objective. Another example is Sukhbaatar et al. (2018), which utilizes unsupervised self-play for learning goal-embeddings.

Florensa et al. (2017) used a Stochastic Neural Network (SNN) (Tang and Salakhutdinov, 2013; Radford, 1990) combined with an information-theoretic regularizer during a pre-training phase in order to discover diverse sub-behaviors. SNNs are able to model stochastic processes, by integrating stochastic units in the computation graph. These stochastic units are used to model a diverse set of sub-behaviors, while non-stochastic units are used to share information across sub-behaviors.

The various sub-behaviors can be activated by feeding an additional extra input to the policy. Different latent codes generate different interpretable sub-behaviors. After pre-training sub-behaviors, a high-level policy is trained, keeping the weights of the sub-behaviors frozen. The higher level selects a sub-behavior through the latent variable and commits to it for a fixed amount of steps. TRPO (Schulman et al., 2015) is used to train both the manager and the lower level.

The Variational Intrinsic Control (VIC) (Gregor et al., 2016) algorithm tries to discover as many sub-behaviors as possible, while simultaneously maximizing the diversity of sub-behaviors. VIC optimizes an empowerment objective (Salge et al., 2014). Empowerment optimizes sub-behaviors to reach states where the agent expects to achieve the most control after learning. This is an unsupervised method for sub-behavior discovery because empowerment is not directly related to the

overall intention (maximizing extrinsic reward) of the agent.

$$\underset{s_0 \sim \mu}{\mathrm{E}} \left[\underset{\tau \sim \pi(\cdot|s_0)}{\mathrm{E}} \left[\log P_D \left(c | s_0, s_T \right) \right] + H \left(G \left(\cdot | s_0 \right) \right) \right] \tag{3.2}$$

The diversity of VIC sub-behaviors is achieved by maximizing the number of different states an agent can easily reach. This can be measured by the mutual information between the set of action choices of a sub-behavior, and the set of termination states. The intuition used is that we should be able to tell intrinsically different sub-behaviors apart if we can infer them from final states.

VIC however is difficult to use in a high-dimensional state space, because function approximation is difficult due to the unstable empowerment intrinsic reward. In addition, exploration is complex, if a new state is discovered, there is probably not yet a sub-behavior, which takes the agent to this new state, so inferring what sub-behavior leads to this state is not yet possible.

Diversity is All You Need (DIAYN) (Eysenbach et al., 2019) also maximizes an information-theoretic objective using a maximum entropy policy. This objective is used to create a set of sub-behaviors that are as diverse as possible. The objective can be interpreted as maximizing the discriminability between different sub-behaviors. This is achieved by maximizing the mutual information between all states of a single trajectory, and the sub-behavior. According to DIAYN, a sub-behavior should control which states the agent visits. Thus, it should be possible to infer the sub-behaviors from the states visited. The behaviors that emerge in this way have been shown to represent various sub-behaviors such as walking and jumping.

$$\underset{c \sim G}{\mathrm{E}} \left[\underset{\tau \sim \pi, c}{\mathrm{E}} \left[\sum_{t=0}^{T} \left(\log P_D \left(c | s_t \right) - \log G(c) \right) \right] + \beta \mathcal{H}(\pi | c) \right] \tag{3.3}$$

DIAYN extends upon VIC in order to be applicable in more complex environments. The distribution of sub-behaviors is fixed in DIAYN rather than learned. This is done in order to prevent a collapse of diversity. VIC learns the distribution over sub-behaviors, which leads to oversampling of already diverse sub-behaviors. Because DIAYN utilizes a uniform distribution, training time is better divided.

Once discovered, sub-behaviors are used for the entire episode. Solving a problem consists of selecting the right sub-behavior. When presented with a problem, DIAYN tests all sub-behaviors and picks the one resulting in the largest reward. Experimentally DIAYN finds sub-behaviors that are able to solve sparse complex benchmark tasks. It even is able to learn multiple alternative solutions for solving a single task.

Achiam et al. (2018) proposed an algorithm that combined option discovery methods with variational autoencoders (Kingma and Welling, 2014). The result is the Variational Autoencoding Learning of Options by Reinforcement (VALOR) architecture. VALOR samples random vectors called contexts from a noise distribution. These contexts are used as additional input to the policy to form trajectories. Secondly, an autoencoder is trained to decode contexts from trajectories. Contexts should become associated with trajectories as a result of training. The VALOR approach was able to distinguish between trajectories meaningfully. Curriculum learning (Bengio et al., 2009) is used, by increasing the number of possible contexts, when the performance on the current set of contexts is strong enough.

$$\mathop{\mathrm{E}}_{c\sim G}\left[\mathop{\mathrm{E}}_{\tau\sim\pi,c}\left[\log P_D(c|\tau)\right]+\beta\mathcal{H}(\pi|c)\right] \tag{3.4}$$

VIC and DIAYN can be considered specific instances of VALOR. VIC, DIAYN, and VALOR all achieved similar performance (Achiam et al., 2018), however, VALOR is able to qualitatively discover better sub-behaviors because of its trajectory focus.

The Latent space policies (LSP) (Haarnoja et al., 2018a) architecture is a multi-level architecture, in which a latent variable of a lower-level layer acts as the action space for a higher-level layer. The intuition is that a layer should either directly try to solve the overall problem or make the problem easier to solve for the next layer. The higher layer can always undo any transformation of the lower layer. This is possible because each layer has access to the state observation and the usage of bijective transformations.

Each layer is trained in turn starting from the lowest level, after training a lower level the weights of this lower level are frozen. Additional layers can then be iteratively trained, using the latent variable of the level below it, as its action space. Each layer is trained using a maximum entropy intrinsic reward. However, for more challenging tasks, this approach also allows incorporating prior information, in the form of a shaping reward. For example, an additional reward for movement could be used at a lower level, in combination with the entropy intrinsic reward. This will lead to movement in all directions. It is possible to use a different reward function for each layer. Training within a layer is done using Soft Actor-Critic (SAC) (Haarnoja et al., 2018b). This approach achieved the best results when trained in a bottom-up fashion by pre-training sub-behaviors, and stacking additional layers on top of the developed sub-behaviors. Training multiple layers, simultaneously end-to-end, reduced overall performance.

Unsupervised methods have proven to be capable of discovering diverse sets of sub-behaviors without any dependency on expert knowledge. However, as the dimensionality of the state space increases, purely using information-theoretic objectives often leads to a trivial encoding of the context (Achiam et al., 2018; Vezhne-

vets et al., 2017). It remains an open question how information-theoretic methods can be diverse in more meaningful ways.

3.7.4 End-to-End Algorithms

Unsupervised discovery of sub-behaviors allows goal-conditional algorithms to discover a wide range of diverse sub-behaviors. While this is especially interesting in a lifelong learning setting (Silver et al., 2013), finding a solution to a single control problem might not require a large set of diverse sub-behaviors.

Algorithms such as FeUdal Networks (FuN) (Vezhnevets et al., 2017) and HIerarchical Reinforcement learning with Off-policy correction (HIRO) (Nachum et al., 2018b) are capable of learning diverse sets of sub-behaviors and their composition, end-to-end, in the function of the extrinsic reward.

FuN (Vezhnevets et al., 2017) is inspired by the Feudal-approach (Dayan and Hinton, 1993), in the sense that a higher-level manager, working at a lower temporal resolution, selects a subgoal, and a lower-level worker, is tasked to achieve this subgoal. Differently from the original Feudal approach, FuN does not use reward hiding, and balances intrinsic and extrinsic rewards through an environment-specific hyperparameter.

The communicated subgoals are defined in a latent space, and they are directional rather than absolute in nature. It is much more feasible for a worker to move the agent in a certain direction than to navigate directly to a subgoal state. Thus, a directional subgoal specification allows a much denser intrinsic reward signal. It was demonstrated that directions in a latent space allow the representation of diverse sub-behaviors. The worker can be trained to maximize this intrinsic reward using any deep RL algorithm. Advantage Actor Critic (A3C) (Mnih et al., 2016) was used by the authors.

The manager learns to select latent goal directions, directly maximizing the extrinsic reward using an approximate transition policy gradient. This form of policy gradient learning exploits the fact that the behavior demonstrated by the worker will ultimately align with the goal set. The manager learns to set advantageous goal directions.

Both the manager and the worker use recurrent networks. The worker uses a standard LSTM network (Hochreiter and Schmidhuber, 1997) while the manager uses a novel dilated-LSTM, which efficiently allows the manager to operate at a lower temporal resolution.

The HIerarchical Reinforcement learning with Off-policy correction (HIRO) (Nachum et al., 2018b) architecture takes a similar approach as FuN, in that it commu-

nicates subgoals between layers in a directional fashion. However, HIRO does not use a latent space to represent the subgoal but is able to actually use the state space to select different sub-goals. Similarly to FuN, the lower level is densely rewarded for moving toward this subgoal state. Using a low-dimensional latent space in order to represent sub-behaviors reduces the number of sub-behaviors that can be expressed. An examination of the impact of the goal representation in terms of the impact of expected reward has been conducted by Dibya et al. (2019) and Nachum et al. (2018a).

The HIRO approach focuses on sample efficiency by supporting off-policy learning. In the case of HIRO, off-policy corrections are used to make up for combinatorial effects introduced by simultaneously learning a lower-level policy and a high-level policy. This correction re-labels past experience with a high-level action, in order to maximize the probability of the past lower-level actions.

A combination of the off-policy correction and a representation of subgoals in the raw state space experimentally showed significantly increased performance over FuN (Vezhnevets et al., 2017), and state-of-the-art hierarchical bottom-up approaches (Florensa et al., 2017). However, stabilizing end-to-end goal-conditional algorithms such as HIRO and FuN remains difficult, and requires additional research.

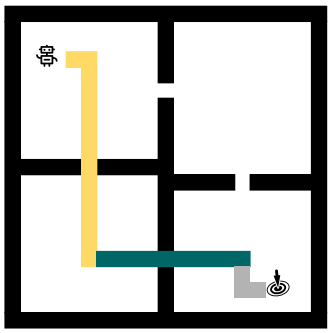
3.8 Benchmarks

In order to compare the performance of different architectures, various benchmark environments have been proposed. In order to qualify as a suitable benchmark, a problem needs to be (Bellemare et al., 2013) varied and interesting enough in order to claim generality and represent a real-world problem. Ideally, benchmark tasks are created unrelated to specific algorithms or research directions in order to avoid experimenter's bias. In this section, we present a representative subset of such environments.

3.8.1 Low-Dimensional State Space Environments

A set of exemplar control problems has been heavily used in order to demonstrate the capabilities of HRL systems. These low-dimensional environments often act as a first testing ground during experimentation. Because of their limited low-dimensional state and action spaces, algorithms often converge quickly, which allows fast iteration.

For example, the four-room grid world (Sutton et al., 1999) is a popular bench-



Figuur 3.7: **Example four-room *grid world***
The agent needs to navigate through the different rooms in order to reach a goal state. Navigation between rooms in this example is done by the yellow and green sub-behaviors. In order to reach the goal state, the agent also needs to use primitive actions, which are displayed in gray.

mark task for hierarchical systems. In this environment, the agent needs to navigate to different locations in four rooms connected by narrow hallways. Often different sub-behaviors are learned to efficiently navigate between different rooms. An example is presented in Figure 3.7.

Another classic discrete action-space task is the Taxi-domain proposed by Dietterich (2000). In this domain, a virtual taxi needs to pick up customers and drop them off at the right locations. More recently an escape-room (Menashe and Stone, 2019) environment has been proposed in order to test hierarchical agents. The objectives in these environments are configurable in difficulty. This configurability allows for a gradual advance to more complex problems than the Taxi-domain, without taking too big steps.

Classical continuous state space environments such as the Cartpole environment (Barto et al., 1983), and the Pinball-domain (Konidaris and Barto, 2009b) are often used to demonstrate algorithms capable of handling a continuous action space.

3.8.2 High-Dimensional State Space Environments

3.8.2.1 Discrete Action Spaces

Video games are an ideal environment for testing HRL algorithms. Gathering large amounts of data from game environments is in most cases inexpensive, and safe. Classic well-known games are often used, because they represent various difficult tasks, and do not suffer from experimenter bias, because they were not explicitly built as an HRL benchmark task.

A set of Atari games (e.g., *Asteroids*, *Breakout*, *Pong*) (Bellemare et al., 2013) has been the most widely used benchmark to demonstrate the capabilities of hierarchical algorithms. Especially Atari 2600 games which are considered hard-exploration games such as *Montezuma's Revenge*, *Pitfall* and *Private Eye* are considered suitable HRL benchmarks, as non-hierarchical methods often struggle to find optimal policies for these environments. HRL algorithms have made significant progress (Vezhnevets et al., 2017; Kulkarni et al., 2016a) on these problems.

While HRL algorithms often score well in some types of games, they are rarely able to score well in all types of games simultaneously. Generalization over multiple games remains an ongoing challenge. In order to reduce overfitting, procedurally generated game environments are often used (Cobbe et al., 2019). By using generated environments the agent has not seen before, we can test whether the agent really mastered the problem, or relied on very specific problems of the environment.

Semi-realistic 3D worlds are also often considered when reporting on the performance of HRL algorithms. For example, *ViZDoom* (Kempka et al., 2016), is based on the first-person shooter Doom. The DeepMind Lab (Beattie et al., 2016) platform provides various challenging 3D navigation and puzzle-solving tasks.

The open-world game of Minecraft (Johnson et al., 2016) is another environment often used to demonstrate complex behaviors of various hierarchical systems. Navigating this environment requires both visual cognition of a high-dimensional environment and planning actions on a higher level of abstraction.

Real-time Strategy (RTS) games have also been used a lot in AI research. Especially, the StarCraft (II) game, has been heavily used as a platform to demonstrate the progress of AI systems (Vinyals et al., 2017; Santiago et al., 2013). While classic Atari games often only have a low branching factor, RTS games typically exhibit very high branching.

With the introduction of various rearrangement tasks (Section 2.6.2.6) within the Habitat 2 simulator (Szot et al., 2021), Habitat now supports the development of hierarchical policies within the framework itself.

3.8.2.2 Continuous Control

In order to demonstrate the capabilities of HRL algorithms in control problems with continuous action spaces, three major virtual environments are commonly used: the MuJoCo (Todorov et al., 2012) framework, the DeepMind Control Suite (Tassa et al., 2018) and the PyBullet simulator (Coumans and Bai, 2016).

In these virtual environments, the agent is tasked with learning the locomotion of

different bodies. The action space typically consists of the amount of torque the agent can apply on various motors. The state space is often made up of different positions in a 3D space. A commonly adopted benchmark in this area is the set of benchmark tasks defined by Duan et al. (2016a). Especially interesting are the two tasks which are hierarchical in nature. In these tasks, the agent is required to learn the locomotion of a body with numerous degrees of freedom, together with navigating various environments. In these control problems, the agent simultaneously needs to be capable of reasoning where to go in the environment, and how to control the actuators in order to move at all. This requires the agent to make decisions on various temporal scales.

3.9 Comparative Analysis

In the following two sections, we provide a comparative analysis. We start this analysis by summarizing the reviewed frameworks first (Section 3.9.1). In the second subsection (Section 3.9.2), we go deeper and compare core features of key HRL algorithms.

3.9.1 Frameworks Summary

In Table 3.1, we provide a short overview of the capabilities and challenges of the different frameworks.

Problem-specific models demonstrated the capabilities of HRL, but proved to have only limited application, mainly due to their dependency on expert knowledge.

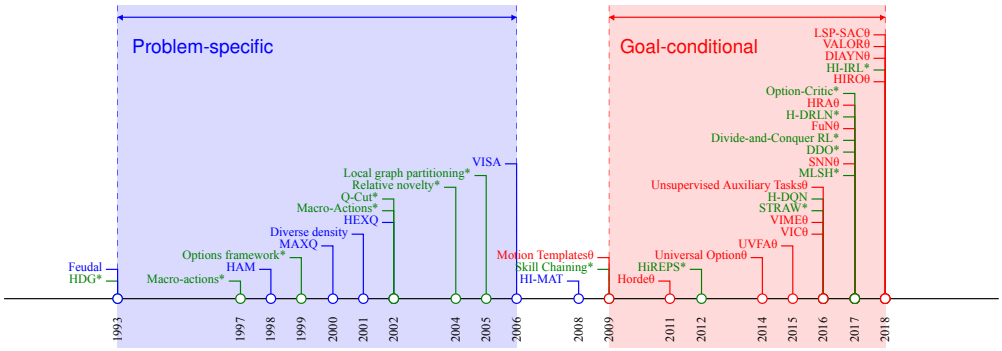
The options framework provides a generic and comprehensive way to model and train a limited number of reusable temporal abstractions. This framework allows various ways to incorporate expert knowledge (e.g., intrinsic reward, termination condition, or state abstractions), while also being generic enough to support the automatic development of sub-behaviors.

The goal-conditional framework, in turn, provides an answer on how to efficiently scale to a larger number of expressible sub-behaviors. Unfortunately, this scalability makes training more unstable, is more difficult to re-use, and produces less interpretable results. It also remains mainly unclear how to efficiently sample different goal vectors in the goal-conditional framework.

As seen on the timeline, presented in Figure 3.8, problem-specific models were mostly used in the early days of HRL. The ability of deep neural networks, to allow RL agents to directly work on high-dimensional state spaces, inspired the

	Problem-Specific Models	**Options**	**Goal-Conditional**
Sub-behaviors	Problem-specific sub-behaviors that work together in order to solve a very specific task.	A more generic system of modules that work together to tackle a limited set of similar sub-problems.	A generic system that is capable of expressing a wide range of sub-behaviors.
Capabilities	Intuitive way of modeling hierarchies of sub-behaviors.	- Generic framework capable of modeling sub-behaviors. - Transfers well in similar environments. - Various learning algorithms available.	- Generalization of sub-behaviors. - Capable of supporting a large amount of sub-behaviors. - Various learning algorithms available.
Challenges	- Not generally applicable. - Requires a lot of expert knowledge. - Lack of learning algorithms.	- Learning is often sample inefficient. - Difficult to share knowledge between options. - Limited scalable: only a few options at the same time are feasible.	- Efficient representation of subgoals. - How to efficiently sample goal-vectors during training, in order to maximize generalization over sub-behaviors. - Often suffers from instability. - Scaling to more than two levels remains difficult.
Required priors	Almost entirely hand-crafted.	Some expert knowledge required in the form of termination conditions, and intrinsic reward signals.	None, however intrinsic reward has been demonstrated to speed up training (Haarnoja et al., 2018a).
Interpretability	High	High, however often due to introduced priors	Low, often uses latent-spaces.

Tabel 3.1: Summary table of the reviewed HRL frameworks



Figuur 3.8: **Timeline of common HRL algorithms**
The problem-specific models (blue) have started interest in the area, but have somewhat lost interest. Goal-conditional (θ, red) algorithms are currently the most common approach, while the options framework ($*$, green) has remained a popular framework throughout the whole HRL history.

rise of a goal-conditional approach. While the options framework has managed to stay relevant throughout the entire short history of HRL, problem-specific models somewhat lost their appeal.

It however should be noted, that most problem-specific frameworks can be modeled using the options framework. And similarly, options can be modeled using a goal-conditional approach. For example, options can be modeled in the goal-conditional framework, by utilizing a discrete goal vector.

3.9.2 Algorithms

The algorithms discussed in previous sections have addressed important issues of RL. Currently, no single algorithm is capable of completely satisfying all our proposed evaluation criteria. In the following subsections, we discuss some trade-offs that exist among current algorithms.

Unfortunately it is very challenging to compare different HRL algorithms on a quantitative basis. This is often caused by the wide range of different used benchmark tasks, and the difficulty of reproducing RL results (Engstrom et al., 2020; Henderson et al., 2017).

In Tables 3.2–3.4 we qualitatively evaluate our selection of key HRL algorithms using the following criteria:

- Training method: how is the algorithm trained? Can it be trained end-to-end, or is a staged pre-training phase required?
- Required priors: how much additional domain knowledge is required, external to the gathered experience in the environment?
- Interpretable elements: what parts of the resulting policies can be interpreted by humans?
- Transferrable policy: are the abstractions proposed by the algorithm transferable to other problems, or are they task-specific?
- Improvement: what is the main improvement of the algorithm over the previous state of the art?

3.9.2.1 Training Method

Deep neural networks have proven to be capable of learning task-specific input features in vision and audio tasks (LeCun et al., 2015). This data-driven approach outperformed previous approaches that rely on a human expert to design input features. RL uses this same data-driven approach to end-to-end learn task-specific input features that will influence the effective behavior of the agent.

One of the goals of HRL is to extend this data-driven approach for RL problems and to provide a temporal module that can be trained end-to-end. Such an approach allows sub-behaviors to be discovered solely in function of the reward signal.

Currently, in all frameworks, there have been algorithms proposed that can be trained end-to-end such as the problem-specific Feudal model (Dayan and Hinton, 1993), Option-Critic (Bacon et al., 2017) in the options framework, and FeUdal Networks (Vezhnevets et al., 2017) in the goal-conditional setting. Unfortunately, it remains challenging for current end-to-end algorithms to discover non-trivial solutions. Solutions often degrade into single-action sub-behaviors, or end up solving the entire task (Vezhnevets et al., 2017). In order to tackle this issue various regularizations have been proposed, such as focusing sub-behavior termination on a small and limited number of states (Harutyunyan et al., 2019), or to add a cost when switching sub-behaviors (Harb et al., 2018).

Alternative to this end-to-end approach is a staged approach in which sub-behaviors are developed independently of the extrinsic reward signal, using a separate pre-training phase. In this approach typically an additional intrinsic objective is used to develop sub-behaviors. This is the most common approach used when develo-

ping options. This intrinsic reward signal could for example be positive when the agent reaches a special termination state (McGovern and Barto, 2001).

In the goal-conditional framework, a staged training approach is commonly used. Because algorithms using the goal-conditional framework are capable of discovering a large number of sub-behaviors, a more scalable information theoretic objective is often used as an additional intrinsic reward signal. This could for example entail a pre-training phase to discover various sub-behaviors leading to a diverse set of states (Eysenbach et al., 2019; Achiam et al., 2018).

Additionally, a pre-training technique often used, especially when dealing with a large number of sub-behaviors, is curriculum learning (Bengio et al., 2009). This technique consists of utilizing a curriculum of different tasks, with increasing difficulty. By compounding knowledge, agents can learn to solve increasingly more difficult tasks (Tessler et al., 2017).

While current end-to-end approaches often lead to trivial solutions, staged approaches are often less sample efficient. In a staged approach, environment interaction is allocated to learning sub-behaviors independently of the extrinsic reward signal, so useless sub-behaviors in terms of the extrinsic reward signal will also be developed.

Some algorithms (Haarnoja et al., 2018a) are capable of end-to-end training, but support pre-training as an optional step.

3.9.2.2 Required Priors

A long-standing trade-off that exists in machine learning, is how much prior knowledge we should incorporate into our algorithms. Incorporating prior knowledge from a domain expert might greatly improve performance, however, it might also lead to unexpected side effects (Ng et al., 1999). Additionally, the environment might be so complex, that it is not possible for a domain expert to specify a formal control policy.

Training agents, using a problem-specific framework, often requires a large amount of prior information. This often includes the entire hierarchical architecture. Unfortunately, hierarchical architectures are often very difficult to be designed by a human expert.

In the options framework, the amount of required prior information is somewhat reduced because of the generic nature of the framework. Typical prior information, required when using option-based algorithms, is the number of options. This value is typically found by conducting a hyperparameter search. Additional forms of prior knowledge that are often required by algorithms that use the option fra-

mework include: intrinsic reward signals, examples of successful trajectories, or a curriculum of tasks with different levels of difficulty.

The goal-conditional framework typically does not require any priors, besides the definition of the goal space. Because this framework is commonly used to discover a large set of sub-behaviors, it would not be feasible to rely on expert knowledge. The goal-conditional framework thus is the framework that most closely matches the goal of being entirely data-driven. Unfortunately, research has shown that being completely data-driven often leads to trivial solutions in complex state spaces (Achiam et al., 2018).

3.9.2.3 Interpretable Elements

One of the benefits of HRL is that a composition of sub-behaviors might be more transparent than one big policy. In the problem-specific setting, algorithms are highly interpretable by nature, however, as discussed before, this expressiveness is often only due to prior expert knowledge.

In the options framework, the sub-behaviors are often interpretable by a human agent because options tend to terminate in states with special properties such as doors or elevators. However, the interpretable elements in an options-based approach, are most often also the elements provided by a human expert.

As the number of sub-behaviors increases, and the amount of utilized expert knowledge decreases, it becomes increasingly difficult to make them transpicuous. A technique that is often used to demonstrate the capacity of an algorithm to discover meaningful sub-behaviors is to plot a number of sampled trajectories from the different sub-behaviors. This rollout technique is often used in order to explain discovered sub-behaviors in the goal-conditional framework.

There however remains a lot of room for further examining the potential of HRL methods, to make RL interpretable. In order to do this, inspiration might be found in how convolutional neural networks are often visualized (Zeiler and Fergus, 2014).

3.9.2.4 Transferrable Policy

Because current HRL algorithms are generally less sample efficient than non-hierarchical algorithms, HRL algorithms are often advertised as being able to transfer a policy well to similar tasks, while non-hierarchical algorithms often need to start from scratch for every new task.

The ability of an HRL algorithm, to efficiently solve multiple tasks, is largely

Algorithm	Training Method	Required Priors	Interpretable Elements	Transferrable Policy	Improvement
Feudal-Q	End-to-end	state space division	None	No, problem-specific solution	More comprehensive exploration than flat Q-learning
HAM-Q	End-to-end	HAM	Trained HAMs	HAM language can be used to transfer knowledge	Significant improvement over flat Q-learning
MAXQ-Q	End-to-end	MAXQ decomposition	MAXQ decomposition	MAX-nodes	Faster training, compared to flat Q-learning
HEXQ	Staged	None	MAXQ decomposition	MAX-nodes	Automatic discovery of MAXQ decompositions
VISA	End-to-end	DBN model	MAXQ decomposition	MAX-nodes	More complex decomposition than HEXQ
HI-MAT	End-to-end	a successful trajectory	MAXQ decomposition	MAX-nodes	More compact hierarchies than VISA

Tabel 3.2: Evaluation of problem-specific algorithms

linked to the method used to train it. Algorithms that use a pre-training stage are better equipped to be used in a multi-tasking setting. This is due to the fact that the lower levels of the hierarchy are trained independently of the task.

Because this staged approach is a popular training approach in both the problem-specific framework and the options framework, these frameworks are currently the most suitable to support an agent with a transferrable policy (e.g., (Frans et al., 2018; Dietterich, 2000)). In these frameworks, the often pre-trained sub-behaviors can be re-used to solve different tasks by learning new higher-level controllers.

The algorithms that fit into the goal-conditional framework unfortunately are currently less capable of transferring their abstractions to new problem settings. This is often due to the problem-specific design of the goal space. Especially when trained end-to-end (Nachum et al., 2018b; Vezhnevets et al., 2017; Bacon et al., 2017), the learned abstractions are fully in function of the problem at hand.

Algorithm	**Training Method**	**Required Priors**	**Interpretable Elements**	**Transferrable Policy**	**Improvement**
Diverse density	Staged	Number of options, successful trajectories	Subgoal states	Same environment	Automatic subgoal detection learns faster compared to using only primitive actions
h-DQN	Staged	Subgoals as pixel masks	Subgoals	Same environment	First HRL approach to hard-exploration in high-dimensional state spaces
HiREPs	End-to-end	Number of options	None	No, problem-specific solutions	Improved performance over non-hierarchical REPS algorithm
STRAW	End-to-end	None	Macro-actions	No, problem-specific solutions	Improved performance in some Atari games
H-DRLN	Staged	Task curriculum	None	Same environment, similar tasks	Demonstrates building blocks for lifelong learning framework
Eigen-Options	Staged	Number of options	Subgoals	Same environment	Discovered options allow better exploration then bottleneck-based options
MLSH	Staged	Number of options	None	Transfer possible to tasks different from previously seen tasks	Faster training performance when applied on new tasks
DDO	Staged	Demonstrations	None	Solutions are task-specific	Faster training in Atari RAM environments.
Option-Critic	End-to-end	Number of options	Termination probabilities	Same environment	First end-to-end algorithm

Tabel 3.3: Evaluation of option algorithms

Algorithm	**Training Method**	**Required Priors**	**Interpretable Elements**	**Transferrable Policy**	**Improvement**
VIC	Staged	None	None	Same environment	Demonstrated capabilities of unsupervised learning in HRL
SNN	Staged	Proxy reward	Sub-behaviors	Same environment	Increased expressiveness and multimodality of sub-behaviors
FuN	End-to-end	None	None	Task-specific solutions	Significant improvement over Option-Critic
DIAYN	Staged	None	None	Same environment	Discovers more diverse sub-behavior than VIC
SAC-LSP	Staged and End-to-end	None, however reward shaping is supported	None	Same environment	Outperforms previously proposed non-hierarchical algorithms
HIRO	End-to-End	None	None	Same environment	Outperforms FuN, especially in sample efficiency
VALOR	Staged	None	None, sub-behaviors are encoded by a latent vector	Similar environments	Qualitatively better sub-behaviors than DIAYN and VIC

Tabel 3.4: Evaluation of goal-conditional algorithms

3.10 Open Research Challenges

Our comparative analysis (Section 3.9) demonstrates, that while great progress has been made in a lot of key areas, there still remain a lot of unanswered questions. In this section, we list some key open challenges in HRL. In order to further advance HRL, we also identify possible moves forward.

3.10.1 Top-Down Hierarchical Learning

Currently, most algorithms in HRL work bottom-up. The agent first learns some sub-behaviors, while exploring the environment. Once these sub-behaviors are sufficiently developed, an agent learns how to compose these sub-behaviors in order to solve complex problems.

This approach however is limited, because it wastes time, learning sub-behaviors, which the agent possibly does not need to solve the task at hand. A top-down approach, which decomposes the problem first, and learns different sub-behaviors in function of this decomposition, is more sample efficient. However, this approach currently tends to lead to unstable learning (Haarnoja et al., 2018a; Vezhnevets et al., 2017; Bacon et al., 2017). Because the transition function of the higher level becomes non-stationary, the higher level will need to take into account that outcomes of sub-behaviors might change, while in a bottom-up approach, the sub-behaviors are typically frozen after development.

Algorithms capable of dealing with non-stationary transition functions could make HRL more sample efficient by enabling top-down learning. A model-based approach could help the agent to test out various subgoals without having to worry about how to reach each subgoal. However, how to efficiently learn such a model from experience remains an open question. Other possibilities in order to tackle top-down learning have been proposed under the form of transition policy gradient (Vezhnevets et al., 2017), in which the higher-level can be trained without the lower level, and the usage of off-policy corrections (Levy et al., 2019; Nachum et al., 2018b). However, additional research in order to stabilize and scale these methods is required.

3.10.2 Subgoal Representation

An important open problem in the framework of goal-conditional hierarchical learning is the issue of goal representation. Various forms of representation have been researched, ranging from the full state space (Nachum et al., 2018b), to more compact representations (Vezhnevets et al., 2017), learned end-to-end. Using the

full state space will allow all states to be reached by at least a single sub-behavior, but this approach is difficult to scale. While alternatively, using smaller representations will reduce the number of different sub-behaviors which can be expressed. Nachum et al. (2018a) examined the relationship between the representation and its ability to represent an optimal policy as a reward-driven optimization problem. This resulted in an impressive performance on some hard high-dimensional, continuous-control tasks.

Another issue regarding representation learning is its ability to generalize over different environments. An effective goal representation should be capable of expressing subgoals in multiple similar environments. When drawing inspiration from human intelligence, we do not plan in the raw state space of the world but use meaningful abstractions. An optimal goal representation will most likely also be lower-dimensional than the full state space, and make use of meaningful abstractions. However, how to learn these meaningful state abstractions, only using interaction gathered from the environment is an open problem.

3.10.3 Lifelong Learning

One of the main promises of HRL is that it should be capable of facilitating the reuse of sub-behaviors. This re-usability of sub-behaviors should both facilitate transfer-learning of new tasks and should allow the agent to solve more challenging issues by extending upon its existing knowledge.

Utilizing sub-behaviors in order to solve similar problems in the same environment is often experimentally demonstrated in research (Frans et al., 2018; Bacon et al., 2017). However, the transferability of sub-behaviors in order to tackle similar problems in different environments is often beyond the current capabilities of agents. It also remains unclear how sub-behaviors can be utilized in order to extend and manage knowledge and to become more efficient at solving ever more complex problems, without suffering from catastrophic forgetting (Kirkpatrick et al., 2017).

The options framework seems a promising direction to handle lifelong learning. This can be implemented by adapting the options framework to allow knowledge management so that the agent can make deliberate decisions about which options to keep, which ones to remove, and which are suitable candidates to be extended. This idea has been explored before (Tessler et al., 2017; Brunskill and Li, 2014), however, an automated approach remains beyond current capabilities.

The generalization capabilities of the goal-conditional framework would also be a great candidate to be facilitated in a lifelong learning context, however how to incorporate knowledge management in the goal-conditional framework has not been sufficiently studied yet.

3.10.4 HRL for Exploration

The options framework has received a lot of research attention, and a lot of automatic subgoal-discovery methods have been proposed (Bacon et al., 2017; McGovern and Barto, 2001). However, these methods suffer from two major limitations. They often only work in certain types of environments, and they are often very sample inefficient, requiring an extensive pre-training phase.

Option learning can be made more efficient by, instead of using a long pre-training phase, to use an incremental exploration approach. After a short exploration phase, options can be formulated which will allow the agent to jump-start the quest to find even better options, by exploring the state space, in a more structured way.

Additionally, alternative forms of intrinsic reward signals, which are not specific to certain environments, such as (episodic) exploration bonuses (Savinov et al., 2019; Burda et al., 2018a), have not yet been fully researched in the context of automatic option discovery but could lead to more sample-efficient option-discovery methods because of their often more dense nature.

The empirical study of Nachum et al. (2019), demonstrated that current HRL algorithms achieve their improved performance on complex tasks, mainly due to temporal exploration (e.g., instead of exploring the outcome of a single random action, take multiple random actions when exploring). They hint at further researching temporal exploration, as exploration strategies for both flat and hierarchical RL agents. An example of such an approach is (Jinnai et al., 2020) which focuses on using options in order to speed up exploration in high-dimensional state spaces by discovering underexplored regions of the state space and developing options that reach those regions.

3.10.5 Increasing Depth

Currently, most HRL approaches (Eysenbach et al., 2019; Nachum et al., 2018b; Bacon et al., 2017; Vezhnevets et al., 2017), make use of a two-layered approach. In this approach, a manager chooses what sub-behavior to activate. Sub-behaviors are then responsible for deciding on primary actions. These two layers work on different levels of temporal abstraction. This is similar to how small companies operate: the owner defines the vision of the company, while the staff executes this vision. However, in order to scale a company, additional management layers are often introduced, allowing different levels to focus on different levels of business abstraction. This architecture is also similar to how multi-level convolutional neural networks are capable of learning complex hierarchies of discovered features.

Algorithms in HRL will also benefit from scaling up to using more than two levels of temporal abstraction, in order to perform planning on a longer horizon. Existing research (Levy et al., 2019; Haarnoja et al., 2018a; Fox et al., 2017) hints at the benefits of using multiple levels. However, these approaches have been hindered by the extra complexities that are induced by having multiple non-stationary transition functions, when using a top-down approach, and decreased sample inefficiencies when using bottom-up approaches.

An end-to-end approach, capable of learning multiple levels of temporal abstractions, in a sample efficient way, without using any expert knowledge, will allow HRL to tackle problems that require complex long-term planning. However, these kinds of algorithms are beyond the current capabilities.

3.10.6 Interpretable RL

One way to address AI safety, (Amodei et al., 2016) is to dissect the agent in order to make sure that it is incapable of executing harmful behavior. Unfortunately, current RL policies are often completely opaque to researchers. Greydanus et al. (2018) developed a method in order to assess what parts of the state space an RL agent considers when taking decisions. This is a useful exploratory instrument in order to assess whether the agent is not overfitting environmental elements. However, as it is dependent on the evaluation of policy rollouts, it is non-trivial to interpret behavior for complex high-dimensional environments (Atrey et al., 2020). An alternative approach consists of Rupprecht et al. (2020), which utilized a generative model over the state space in order to generate example states that adhere to user-specified behaviors.

HRL will be helpful to further advance this relativly under-explored area (Shu et al., 2018), as a single complex behavior, might be too incomprehensive to safely deploy in the real world. HRL techniques could potentially be used to develop sets of sub-behaviors that can be comprehensive by themselves. Alternatively, HRL algorithms might be used to split an existing policy into multiple smaller, more comprehensive parts.

3.10.7 Benchmark

As discussed in Section 3.8, a lot of different benchmarks are used today, when presenting HRL research. Additionally, researchers often have trouble reproducing presented results (Henderson et al., 2017), due to the complex nature of many RL algorithms.

This often makes it difficult to assess presented results in terms of their predecessors. A novel benchmark, or a standardized set of tasks using existing benchmark environments (such as those proposed in (Duan et al., 2016a)), capable of assessing the qualities of algorithms, to learn sub-behaviors, in a sample efficient way, in order to solve a range of different tasks, will allow better comparability of reported results. As curriculum learning (Bengio et al., 2009), seems to be an important element of many HRL approaches, the difficulty of this novel environment should be tunable. While current benchmark tasks often require a mix of different capabilities, it might benefit the field to propose tasks, which zone in on a single capability (e.g., memory or exploration) (Osband et al., 2019).

Additionally, a suite of open-source high-quality baseline implementations of HRL algorithms, similar to the OpenAI baselines library (Dhariwal et al., 2017), will allow researchers to benchmark their own algorithms reliably.

3.10.8 Alternative Frameworks

Early models, such as MAXQ and HAM, have somewhat lost interest in the research community due to the lack of automatic learning methods, and their dependence on expert knowledge. Interest has shifted to more generic frameworks, for which different learning algorithms have been proposed. However, due to this shift, to more generic frameworks, algorithms have become opaque to humans again.

It might be interesting to revisit these early models and incorporate knowledge gained while researching other frameworks, in order to come up with automated learning algorithms for these methods. It might for example be interesting to research their ability to handle high-dimensional state and action spaces using approximation methods. An example of such serendipity is the recent, goal-conditional, FeUdal Networks architecture (Vezhnevets et al., 2017), which has been inspired by, Feudal-Q (Dayan and Hinton, 1993), one of the earliest problem-specific frameworks.

Alternatively, novel ways of incorporating expert knowledge in the goal-conditional framework have emerged. A noteworthy example (Jiang et al., 2019) utilizes the compositional nature of natural language instructions.

As the options and goal-conditional frameworks also have significant limitations, these frameworks might not be the final answer on how to model hierarchical learning agents. Increasing the depth will allow agents to use more levels of abstraction, but this approach might only scale limitedly. In order to scale to problems, which require very long-horizon planning, without dense feedback, it might be required to look at radical alternative approaches, such as representing sub-behaviors in a graph-structure (Sohn et al., 2020; Shang et al., 2019) or taking

into account relations between objects (Zambaldi et al., 2019; Santoro et al., 2017; Diuk et al., 2008).

3.11 Conclusion

In this chapter of this thesis, the necessary insights to understand the fundamentals of HRL are provided. An overview of the most common frameworks in use today was provided. For each framework we reviewed algorithms, we deem essential to the framework. Intuitively, HRL is a viable option, to allow RL to tackle problems, in environments with very delayed, and sparse reward structures. This intuition is supported by various research conducted on each of the frameworks. HRL has been able to make significant progress in various RL benchmarks, in which previously no progress had been made. However, each framework currently also suffers from its limitations: they are either too dependent on the structure present in the environment, not sample efficient, difficult to learn end-to-end without expert knowledge, or unable to scale and generalize. In order to address the current limitations of HRL approaches, we concluded this survey with some ideas on how to tackle these open challenges. In the next chapter, a novel approach will be described that works towards solving some of these limitations.

4

SETIE: Structured Exploration Through Instruction Enhancement

The contributions presented in this chapter are based on the publication titled: *"Structured Exploration Through Instruction Enhancement for Object Navigation"*.

4.1 Introduction

Finding objects in unseen environments is a hard navigation task (Section 2.6.2.3). In order to be successful, an agent needs to be capable of mastering multiple skills:

- The agent needs to be capable to explore the environment in a structured manner: it should figure out the layout of the previously unseen environment, keep a memory of past actions, and remember visited regions.
- The agent needs to be capable to understand the instruction: map an instruction to an actual visual representation.
- The agent needs to be capable to make decisions on multiple abstraction levels: navigating to the other side of the building versus navigating through a doorway.

These problems have been studied individually intensively in various settings (Szot et al., 2021; Weihs et al., 2020; Savva et al., 2019; Chevalier-Boisvert et al., 2019). However, constructing an agent capable of simultaneously performing these feats, remains an open challenge. In this chapter, we study how we can build an agent capable of simultaneously handling long-term planning through abstraction, low-level locomotion, and basic language grounding.

Current navigation solutions typically utilize a *sense-plan-act* approach, in which different modules interact with each other. These solutions however tend to be brittle, are prone to error propagation, and often require a lot of manual engineering (Karkus et al., 2021; Mishkin et al., 2019). End-to-end RL systems have recently been proposed, as an alternative learning-based solution, to handle these issues (Wijmans et al., 2020). Unfortunately, as empirically demonstrated in Section 4.3.2, RL agents are often unable to reason on multiple levels of abstraction, have difficulties with mapping language instructions, and often explore poorly.

In contrast, the proposed Structured Exploration through Instruction Enhancement (SETIE) approach allows the agent to reason and explore on multiple levels of abstraction (e.g., on room-level and actuator-level) through utilizing a hierarchical approach (Chapter 3). This exploration scheme can be seen as an answer to the exploration challenge defined in Section 3.10.4. The proposed agent can be trained using only the reward signal received from the environment and only requires a simple training curriculum.

In order to work towards solving the challenge of subgoal representation defined in Section 3.10.2 we introduce *instruction enhancements* in order to communicate between the different layers. In this system, the top level is allowed to enhance the instruction it received from the environment. For example, if the original instruction is: *"Find the red ball"*, the top-level might choose to enhance this instruction to: *"Find the red ball, in the kitchen"*. This allows the top level to plan on a higher level of abstraction (Which room makes sense to visit next? Where have I already been?). In turn, the enhanced instruction makes the task more tractable to complete by the lower level.

Because both traditional and learning-based approaches are still unsolved, we take one step back from the typically used photo-realistic simulators (Szot et al., 2021; Savva et al., 2019), and utilize a visually simpler setting, while keeping most of the navigation and generalization complexities. In this setting, it is demonstrated why a flat, non-hierarchical RL agent, does not manage to make any progress, and how the proposed hierarchical approach is capable of exploring the environment in a more principled way. The generalization capabilities of the agent are demonstrated as well. These capabilities allow the agent to find previously unseen objects in new unseen environment configurations.

The contributions of SETIE are three-fold:

- A novel dual-layer hierarchical approach, capable of simultaneously learning structured room-level exploration and low-level navigation.
- In order to communicate between layers, the idea of instruction enhancement is proposed, allowing loose coupling of layers and generalization to novel instructions.
- The introduction of a goal assessment module, which is capable of addressing whether the current state satisfies the instruction, and thus allows offloading language grounding, and integration of prior knowledge in a learning-based setup.

4.2 Approach

The proposed novel approach consists of three parts:

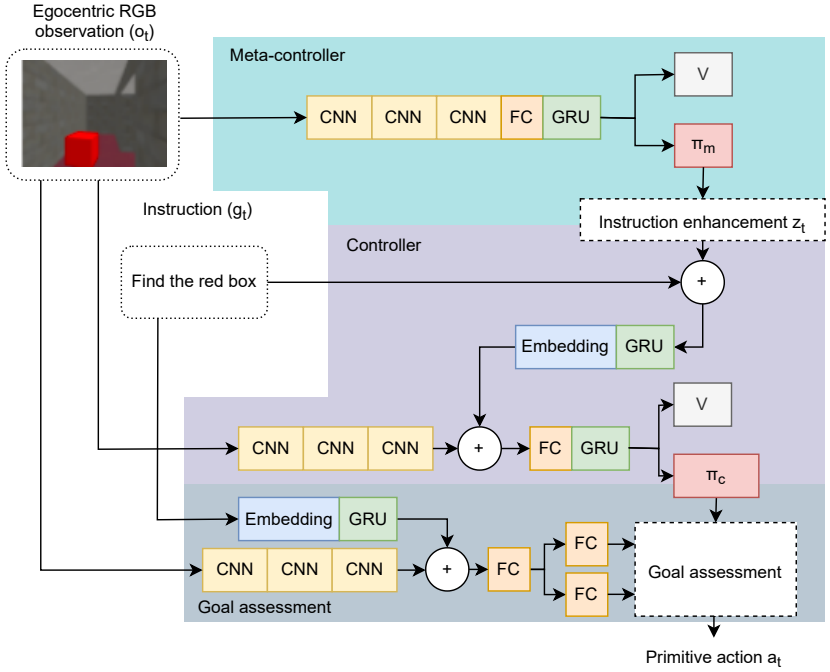
- **The meta-controller** $\pi_m(z_t|o_t, g_t)$ which performs high-level planning, by working on a lower temporal resolution.
- **The controller** $\pi_c(a_t|o_t, g_t, z_t)$ which handles low-level navigation.
- **The goal assessment module** $G(o_t, g_t) \rightarrow \{1, 0\}$ which handles language grounding.

A visual representation of the architecture is displayed in Figure 4.1.

4.2.1 Meta-controller

The meta-controller $\pi_m(z_t|o_t, g_t)$ is responsible for learning high-level navigation of the environment solely from partial state observations (through an egocentric RGB camera). This task consists of two sub-tasks:

1. Discovering the layout of the current environment, determining which rooms are connected to which other rooms. Commonsense reasoning (the garage is less likely to be connected to the bathroom) together with a trial-and-error approach can be used in order to solve this task.
2. Keeping an implicit memory of which rooms have already been visited in order to explore the environment in a structured manner. Because the meta-controller operates on a higher level of abstraction, the agent is capable of performing these tasks using a generic GRU component (Cho et al., 2014) in its architecture.



Figuur 4.1: **An overview of the SETIE architecture.**
The meta-controller handles structured exploration between different rooms from egocentric observations by enhancing the instruction. This output is used by the controller, in order to return primitive actions (navigation). The goal assessment module is used for language grounding.

The action space of the meta-controller consists of a discrete set of *instruction enhancements*. This set of enhancements is provided up-front to the agent. Instruction enhancements should be defined on a higher level of abstraction, than the primitive actions utilized by the controller. By introducing this additional level of abstraction, the agent is able to explore in a structured manner (e.g., room by room).

The meta-controller does not interact with the environment itself, but can only influence the behavior of the controller through enhancing the instruction. For example the extrinsic instruction g_t could have been *"Find the green key"*, which the meta-controller can enhance to become *"Find the green key, in the dining room"*.

Within HRL, designing a sub-behavior space $\mathcal{Z}$ is a complex challenge (Section 3.4.1). Most often this space is tightly coupled between the different levels. Utilizing language allows decoupling multiple levels. This allows the controller and meta-controller to be trained independently. Furthermore, language has also the potential to generalize to unseen instructions (Jiang et al., 2019) and can make the intention of the agent clear to a human in the loop (Chen et al., 2021).

The meta-controller acts on a lower temporal resolution and is asked to provide a new instruction enhancement every c timesteps.

As the meta-controller has no direct influence on the environment, but only can act through the controller, its training needs to take into account the potential unexpected behavior of a trained controller. Such quirks might be the over-exploration of some rooms, while quickly moving through others. Accounting for these eccentricities can be done by utilizing a fully trained and frozen controller during the training of the meta-controller. In this setting, the meta-controller observes the environment, selects an instruction enhancement, and waits until the controller has taken c-steps, before sampling a novel enhancement. The reward of the meta-controller consists of the discounted sum of the extrinsic reward collected during the usage of the active instruction enhancement:

$$R_t(s_t) = 1/c \sum_{t=0}^{c} \gamma^t r_t \left(o_t, g_t, a_t, o_{t+1}\right) \tag{4.1}$$

A second option to train the meta-controller consists of assuming a perfectly behaving controller. In this setting, the simulator will carry out the enhancements, and move the agent to different rooms while respecting the floor plan. Utilizing this second approach allows both controller and meta-controller to be trained in parallel (as there is no dependency). In order to utilize this second training scheme a different reward function is required. For example, a reward function based on room coverage can be utilized. In this setting, each instruction enhancement that takes the agent to a previously unvisited room will lead to a positive reward (0.1), while other proposed enhancements will result in a negative slack penalty (-0.01).

While in the empirical evaluation of the presented method instruction enhancements consist of rooms to navigate between, other sets of enhancements can be used in different settings.

4.2.2 Low-level Controller

The controller $\pi_c(a_t|o_t, g_t, z_t)$ interacts with the environment through its primitive actions $a_t \in \mathcal{A}$. The controller expects on each timestep an egocentric RGB observation of the environment $o_t \in \mathcal{O}$ together with a task instruction $g_t \in \mathcal{G}$ and an instruction enhancement $z_t \in \mathcal{Z}$ provided by the meta-controller. The instruction informs the agent of its objective (e.g., *find the red ball*), and the instruction enhancement (e.g., *in the kitchen*) adds additional information on how the instruction should be carried out. The instruction enhancement will essentially navigate the agent to different rooms, resulting in episodic exploration of the different rooms in order to solve the main instruction. Both instruction and enhancement are provided using simple natural language sentences.

The action-space $\mathcal{A}$ of the controller consists of a discrete set of primitive movement steps (*move forward, turn left, turn right*) and a special *done*-action. This special done-action is invoked when the agent perceives itself near the goal object. Utilizing this action will typically end the episode.

Due to the utilization of instruction enhancements, the low-level controller can be trained independently of the meta-controller. A straightforward way of training the controller is to enhance the instructions by utilizing an oracle. When this oracle provides the most useful enhancement (e.g., which room should the agent visit next to find the goal) the extrinsic reward signal can be utilized to reward the agent. For example in the setting of object navigation, controllers can be rewarded by utilizing the improvement in geodesic distance between the agent and the goal object.

4.2.3 Goal Assessment Module

To signal that the agent believes it has completed the objective, it needs to use a special *done*-action. Utilizing this action will typically end the episode. However, as empirically demonstrated in Section 4.3.2, incorrect usage of this action is one of the main failure modes appearing prior to the introduction of a goal assessment module.

In order to integrate the goal assessment module, the done action is removed from the action space of the controller. Instead, a *query*-action is added to this action space. This novel query action will not terminate the episode (soft termination)

but will query the goal assessment module. If the goal assessment module deems that the instruction is satisfied, and the agent is close enough to the target object, the agent will utilize the original done-action.

Essentially, the controller is now able to focus on low-level navigation and consult an expert (the goal assessment module) in order to handle the language grounding of the instruction.

In order to allow the agent to find objects it did not see during training, a novel goal assessment model can be trained independently of the controller and meta-controller. This is useful, as training a controller and meta-controller is typically more computationally expensive.

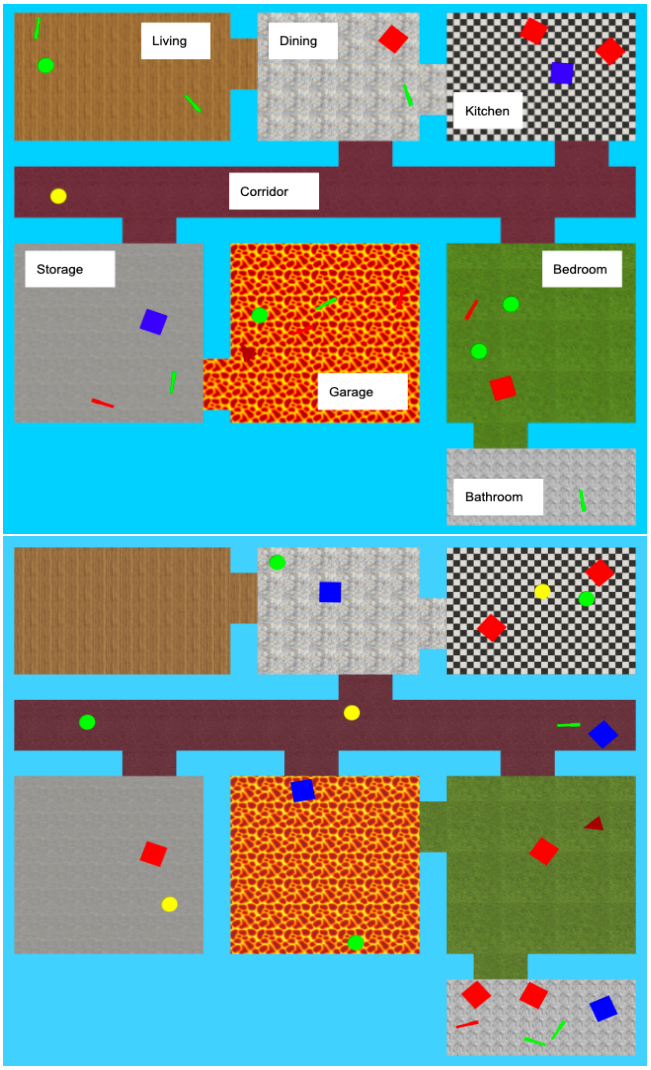
To collect training data for the goal assessment module a random agent can be used, collecting both examples with goal objects, and observations without any visible objects. For positive samples, the correct positive class is utilized 50% of the time, while in the remaining cases, another random possible instruction is utilized, together with the negative class label. This allows for balancing out positive and negative labels.

4.3 Empirical Evaluation

4.3.1 Environment Description

In order to demonstrate the effectiveness of the approach, a simulated typical domestic environment is constructed within the *MiniWorld* framework (Chevalier-Boisvert, 2018). Two instances are represented in Figure 4.2. The environment consists of 7 different rooms (garage, storage, bedroom, bathroom, living room, dining room, and kitchen) together with a corridor that connects some of these rooms (depending on the instance). Each room has a distinctive look. As not all rooms are connected, the agent will often need to backtrack to previously visited points in order to explore the environment.

Throughout the environment, different abstract objects are randomly placed. Objects are defined by a category and a color. The categories used are *box*, *ball*, and *key*. In the experiments, there is typically one goal object and multiple *distractor* objects. In each task instance, there is only a single object which matches the goal object description. The task is communicated using language through the template of *Find the [color] [shape]*. The following objects are used during training: *red box, green ball, blue box, yellow ball, red key, and green key*. There is no association between objects and rooms. While it is definitely also possible to use real objects, the focus of this study is not on object detection. Through utilizing shapes



Figuur 4.2: **Environment used for empirical evaluation of SETIE**
Two different instances of the used environment are rendered. Connections between rooms are randomized (with a holdout set of configurations). The agent has no access to this top-down map view.

and colors we are able to study whether the agent is capable of grounding these concepts.

On each timestep, the agent observes an egocentric RGB observation o_t of the environment. The reward function is densely defined and consists of the improvement in the geodesic distance between the agent and the goal object. We use a slack penalty of 0.01 which is subtracted from the reward on each timestep. When reaching the goal object we award the agent with a success bonus of 10. This reward scheme was proposed in (Savva et al., 2019).

$$r_t(s_t, a_t, g_t) = (-\Delta_{geo\_dist} - 0.01) + 10 * \mathbb{1}_{success} \tag{4.2}$$

Regarding actions, the agent is capable of turning left and right for a fixed amount, moving a fixed distance forward, and utilizing a special *done*-action. In order to successfully complete an episode, the agent needs to use this done-action close to the goal object.

In each episode, the agent starts in a random position and has no access to its current position, the name of the room it is in, or a map of the environment. The connections between the different rooms are randomly enabled. However, each room is always accessible, and there are no uncommon connections (e.g., the bathroom is never connected to the kitchen). In total this results in 132 different possible floor plans. A holdout set of 30 floor plans is not utilized during training but kept solely for evaluation purposes. This holdout set can be used in order to assess the generalization capabilities of the agent regarding floor plans.

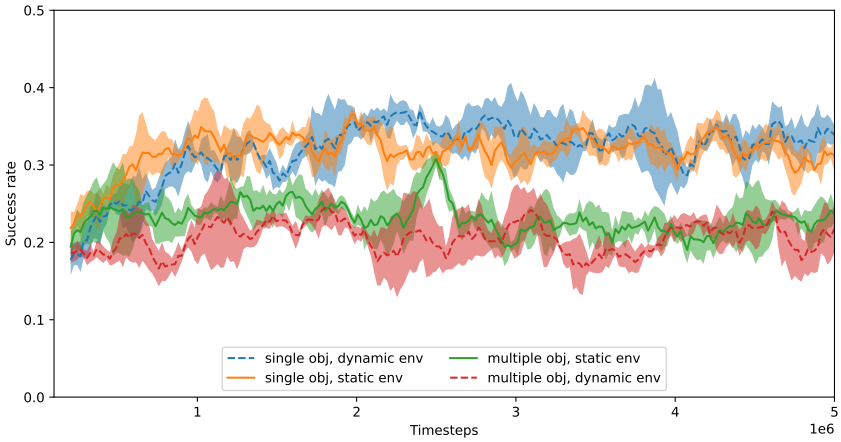
4.3.2 Baselines: why do non-hierarchical approaches fail?

4.3.2.1 With soft-termination

When utilizing a non-hierarchical PPO agent (Schulman et al., 2017) without any instruction enhancements, and with only a single object (a red box or blue box) the agent is capable of achieving an average success rate of $\sim 35\%$ after 5 million interactions with the environment (Figure 4.3). When also introducing the problem of language grounding, by adding multiple objects to the environment, the agent has an average success rate of $\sim 20\%$ after 5 million interactions.

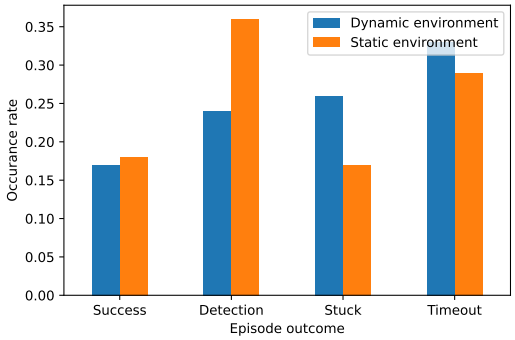
4.3.2.2 No soft-termination (full problem setting)

If we also remove the relaxation of soft termination of the environment, we arrive at the full problem setting. In this setting, when the agent utilizes the *done*-action



Figuur 4.3: Training performance of a non-hierarchical PPO agent with soft-termination. Results are averaged over 3 runs.

incorrectly, the episode is terminated. We analyzed the different failure modes of the baseline agent in this setting (Figure 4.4):

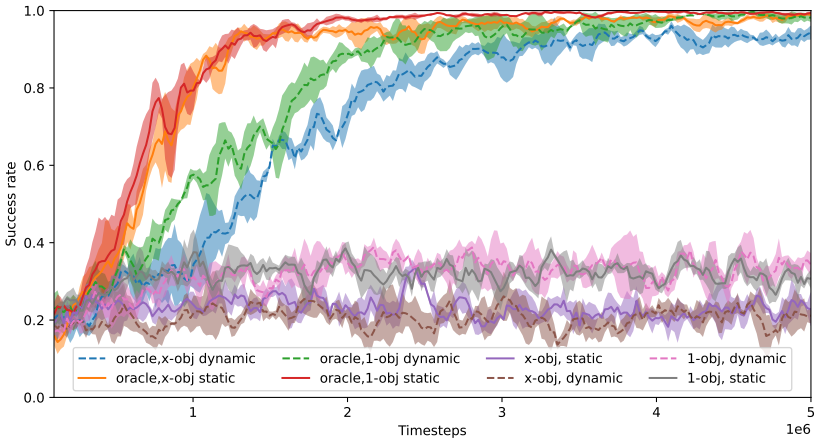


Figuur 4.4: Failure modes of the trained non-hierarchical baseline. If the floor plan remains fixed (static environment), the amount of episodes where the agent gets stuck decreases, however, this in turn increases goal detection errors.

- **Detection:** agent used *done*-action but was in the wrong position.
- **Timeout:** agent did not manage to find the goal within the allowed amount of timesteps, the agent did not use the *done*-action at all.
- **Stuck:** distance between agent and goal object did not change in the final 10 steps.

When looking at these failure modes we noticed that the main reason for failure in a static environment setting is related to the detection of goal objects. When also making the environment dynamic, both local navigation problems (getting stuck), and planning problems (timeout) start to occur more frequently.

4.3.3 Does enhancing the instruction make the task more tractable?



Figuur 4.5: Training performance of the controller, in this setting the agent is allowed to use the done-action multiple times (soft termination). Without information about which room the agent should move to next (oracle), the agent is unable to learn a policy in the environment. Results are averaged over 3 runs.

From the previous section, we can conclude that a non-hierarchical agent is not able to reliably solve the studied task. In order to validate whether enhancing the instruction will improve the performance, we trained an agent with its instructions enhanced through the use of an oracle.

The utilized oracle is aware of the shortest path to the goal object in terms of rooms to visit. Having access to such an oracle outside the training environment

is an unrealistic assumption. The learned meta-controller will, however, take over the role of this oracle, providing adequate enhancements.

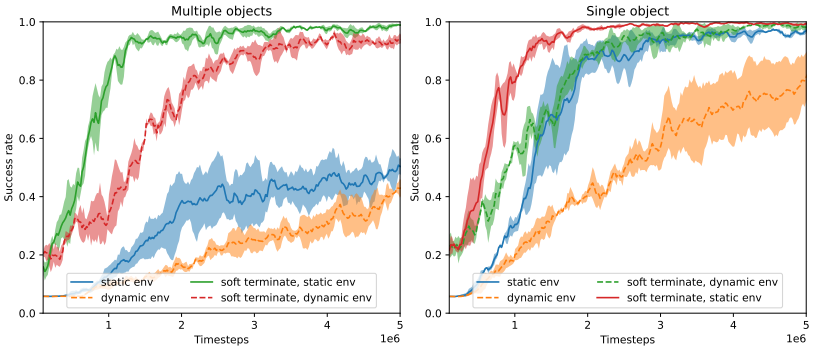
Utilizing an oracle based on the shortest path also alleviates the requirement of a custom reward function. If the controller is able to correctly interpret and follow the instruction enhancement, it will also collect the most reward.

As the results plotted in Figure 4.5 indicate, enhancing the instructions allows the agent to solve most instances of the task both in the setting with a single object (1-obj) and multiple objects (x-obj). This validates the idea that enhancing the instruction allows the controller to carry out the low-level control task.

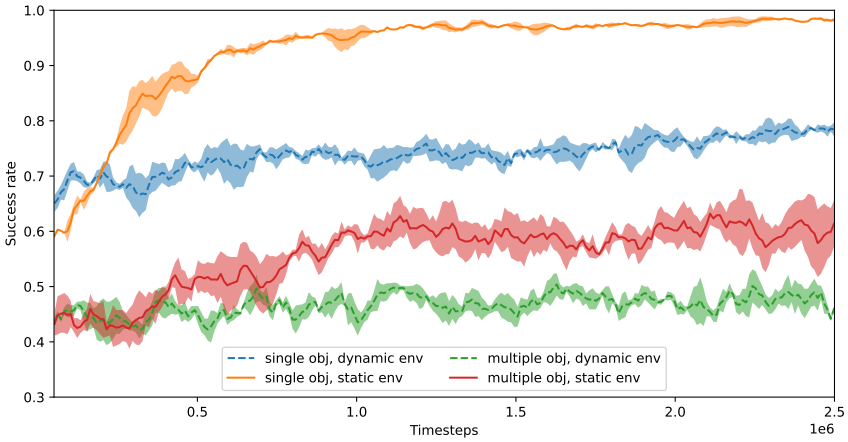
In order to solve the entire task there is still the need to remove soft termination, and actually train a meta-controller.

4.3.4 What is the impact of soft termination?

In the previous experiments, the controller was trained using soft termination. This means that the agent is allowed to use the *done*-action multiple times in an episode. Normally, this would terminate the episode, however, we found that allowing the agent to utilize this action multiple times during training significantly increased the sample efficiency and success rate (Figure 4.6). This can be attributed to the given that the agent is able to collect more negative examples, without being too heavily penalized. This training mechanism is especially crucial in settings that require language grounding (multiple objects). We can allow this constraint due to our goal assessment module, which will filter out invalid done-actions when utilizing the entire architecture.



Figuur 4.6: Training performance of the controller, with oracle instruction enhancements. Allowing soft termination greatly improves sample efficiency. Results are averaged over 3 runs.



Figuur 4.7: Meta-controller training performance. Results are averaged over 3 runs.

4.3.5 Does a trained controller allow the meta-controller to solve the task?

In Figure 4.7 the results from training a meta-controller in various configurations are plotted. The meta-controller has no problem exploring the environment when there is only a single static environment configuration used with a single goal object placed in it (SR $\sim$ 95%). When multiple objects are present in the static environment setting, performance receives a significant hit (SR $\sim$ 60%), but the agent is still able to improve performance.

When the agent needs to manage dynamic instances of environments it starts with a high success rate and is able to steadily improve (SR $\sim$ 70%) in the setting with a single goal object. However, in the setting with both a dynamic environment configuration and multiple objects, the agent is not able to improve its initial performance (SR $\sim$ 45%).

4.3.6 Is the agent capable of exploring in a structured way?

The failure modes of the baseline agent indicated that a lot of episodes ($\sim$ 30%) failed due to the agent running out of allowed steps. This might indicate that the baseline agent is not able to explore the environment in a structured manner. In Table 4.1 we compare the percentage of the rooms the agent visited. From the results plotted in this table, we can conclude that the hierarchical approach is capable of covering a significantly larger proportion of the environment on average.

Agent	Objects	Environment	Room coverage
Hierarchical	Single	Static	51.0%
		Dynamic (holdout)	45.4%
		Dynamic (train)	45.5%
	Multiple	Static	50.2%
		Dynamic (holdout)	36.4%
		Dynamic (train)	36.7%
Flat (baseline)	Static	single	27.8%
		Dynamic (holdout)	25.8%
		Dynamic (train)	26.3%
	Multiple	Static	12.5%
		Dynamic (holdout)	12.6%
		Dynamic (train)	12.6%

Tabel 4.1: Average room coverage observed during evaluation runs

Architecture	Objects	Static	Train	Test
Flat PPO baseline	Single	$37\% \pm 3.71$	$42\% \pm 4.79$	$44\% \pm 3.76$
Hierarchical + GA	Single	$81\% \pm 5.20$	$76\% \pm 5.54$	$75\% \pm 4.67$
Flat PPO baseline	Multiple	$13\% \pm 3.41$	$15\% \pm 4.47$	$12\% \pm 2.66$
Hierarchical + soft term.	Multiple	$82\% \pm 6.44$	$69\% \pm 5.76$	$67\% \pm 5.27$
Hierarchical	Multiple	$15\% \pm 3.12$	$18\% \pm 2.81$	$15\% \pm 3.13$
Hierarchical + GA	Multiple	$52\% \pm 3.06$	$38\% \pm 4.58$	$40\% \pm 4.52$

Tabel 4.2: Overall performance of the entire architecture. For each setting, 10 runs of each 100 random episodes were used.

4.3.7 How well does the proposed hierarchical architecture perform?

In this section, the performance of the architecture is analyzed in its entirety. We are especially interested in how well the agent is capable of handling unseen environment floor plans, and novel objects.

4.3.7.1 Zero-shot transfer to unseen environment configurations

The agent is allowed to utilize 102 different floor plans during training. In order to validate whether the agent is capable of functioning in an environment it did not see during training, there is also a test set containing 30 floor plans the agent did not see during training.

From the results plotted in Table 4.2 we can conclude that the hierarchical appro-

Environment:	**Static**	**Train**	**Test**
Flat PPO Baseline	$15\% \pm 1.69$	$15\% \pm 3.1$	$14\% \pm 4.21$
Hierarchical	$17\% \pm 2.54$	$15\% \pm 2.96$	$14\% \pm 3.52$
Hierarchical + soft term.	$78\% \pm 3.75$	$69\% \pm 2.75$	$66\% \pm 3.77$
Hierarchical + GA	$52\% \pm 4.36$	$39\% \pm 3.7$	$38\% \pm 4.58$

Tabel 4.3: Overall performance of the entire architecture on a holdout set of goal objects. In each configuration, 10 runs of each 100 random episodes were used.

ach has a high success rate in the static environment. Especially, when there is no language grounding required.

In the setting with multiple objects, the hierarchical agent is now able to reach a high success rate when soft termination is allowed. When soft termination is disabled, the goal assessment module is capable of somewhat emulating this improved performance. However, there still remains room for improvement. When qualitatively looking at the mistakes made by the goal assessment module, we noticed that it often made mistakes if the goal object was barely visible in the single-passed RGB observation.

In all cases, the agent was successfully capable of achieving a similar level of performance in the floor plan holdout set as in the training set.

4.3.7.2 Zero-shot transfer to unseen goal objects

Because the instructions are formulated in natural language, we have an interface that makes it straightforward to test how well the agent handles combinations of colors and objects it did not see during training. The goal assessment module was retrained in order to be capable to detect the novel combinations of colors and shapes while keeping all original navigation policies (controller and meta-controller).

Similar to the zero-shot environment transfer experiments, we empirically can validate from the results in Table 4.3 that the agent is able to successfully find combinations of colors and shapes the agent did not see before without having to re-train the controller and meta-controller.

4.4 Conclusion

In this chapter, the problem of structured exploration in an object navigation setting is studied. It is demonstrated how the three sub-problems of navigation, high-level reasoning, and language grounding each contribute to the overall complexity of object navigation. A hierarchical approach is proposed in order to handle both low-level navigation and high-level planning. In order to have a loose coupling between the layers, language is used to enhance the original instruction in a way that makes it feasible for a low-level controller to partially tackle the overall task. To handle the third sub-problem of basic language grounding, a goal assessment module is introduced in order to guide the controller in assessing whether goal objects have been reached.

The effectiveness of the proposed architecture is empirically demonstrated in a simulated domestic environment. It is demonstrated that the agent is able to better handle unseen environment configurations, and unseen goal objects compared to a non-hierarchical baseline.

The focus of this chapter was on the efficient usage of abstractions. More specifically we proposed a solution to the defined challenges of subgoal representation (Section 3.10.2) and structured exploration (Section 3.10.4).

The utilized abstractions were provided up-front, and their policies are learned from scratch. This approach has two major limitations. Firstly, the agent has no method to extend its set of abstractions with novel abstractions when needed. And secondly, developing the policy of each abstraction separately is inefficient. In Chapters 5 to 7 we provide methods that solve these problems (defined in Section 3.10.3) by allowing abstractions to be adapted.

5

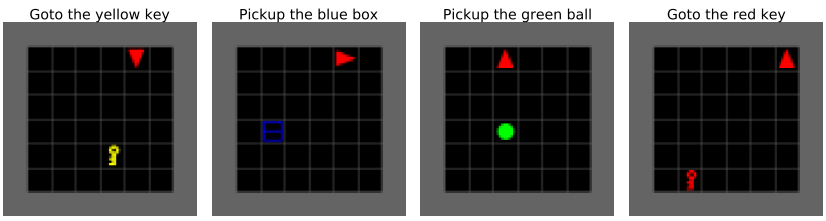
Language Grounded Task-Adaptation

The contributions presented in this chapter are based on the publication titled *"Language Grounded Task-Adaptation in Reinforcement Learning"*.

5.1 Introduction

In the previous chapter, we introduced SETIE, a novel way of solving complex problems through learned abstractions. In this chapter, however, we focus on how we can make the learning of such abstractions more sample efficient. Sample inefficiency within the RL framework is often caused by the reward function specification. On the one hand, a sparse and delayed reward signal makes it difficult for the agent to experience any meaningful feedback. On the other hand, designing tasks with a dense reward signal (reward shaping) is often a complex endeavor, and regularly exhibits unwanted side effects (Ng et al., 1999).

A recent line of research (Luketina et al., 2019), has proposed methods that allow task descriptions to be specified using natural language. A commonly used approach consists of directly embedding both visual observation and language instruction in order to train a policy (Hermann et al., 2017; Misra et al., 2017; Chevalier-Boisvert et al., 2019). Alternatively, (Goyal et al., 2019) uses natural language reward shaping, by predicting if an action in a trajectory matches a task



Figuur 5.1: **Example episodes in the BabyAI environment**
The agent is represented as a red triangle. In each episode, the agent receives the language instruction displayed on top.

description. Jiang et al. (2019) explores the compositional structure of natural language in order to learn abstractions capable of generalizing over different sub-tasks using language instructions. Unfortunately, these methods have proven to still be very sample inefficient, requiring weeks of training in simulations, in order to learn relatively simple tasks.

In this chapter, a natural language-guided transfer learning method is introduced. The proposed method can make RL methods informed by natural language more sample efficient, requiring less interaction with the environment. This is achieved by providing a viable way of allowing an agent to efficiently adapt previously learned knowledge (e.g., in the form of abstractions), to a new previously unseen task. Current algorithms capable of quickly adapting their policies to solve related tasks, mostly rely on intensive training using a diverse set of tasks (Hessel et al., 2019), often guided by a hand-crafted curriculum of increasingly more difficult and diverse tasks. Our method does not require such extensive training and is capable of, given a small set of pre-trained policies, to make decisions about which previously developed policy will adapt best, in order to solve a new previously unseen task, solely from its task description formulated using natural language. The novel proposed method can be seen as an approach to make RL more sample efficient and facilitate lifelong learning through abstraction usage (Section 3.10.3).

5.2 BabyAI Environment

To demonstrate the introduced method, we make use of the *BabyAI environment* (Chevalier-Boisvert et al., 2019). This general platform was proposed in order to study the sample efficiency of grounded language acquisition.

In this environment, the agent is tasked with completing various tasks in a multi-room 2D grid world. For our experiments, we consider a single room and test our method on the *goto* and *pickup* problems. Some example tasks are plotted in Figure 5.1. The task the agent is charged with is described using a synthe-

tic *baby* language. Instructions used in the transfer experiments follow the same $\langle$*verb*, *object color*, *object*$\rangle$ pattern (e.g., *pick up the yellow box*).

The pixels of the screen, together with this instruction, form the fully observable state space $\mathcal{S}$. The action space $\mathcal{A}$ consists of movement, handling objects, and opening doors. The agent only receives a sparse success reward upon completing the entire task.

5.3 Task-Adaptation Method: Sampling Approach

The main idea of the proposed approach is to start with a limited set of pre-trained parameterized base control policies. These policies can be either used to solve the task directly, or they can be linked to an abstraction in a hierarchical setup. Ideally, instead of starting from scratch, we would like to reuse parameters from a prior policy to jump-start a new task. It is however non-trivial to select such a policy. Performing this selection intuitively requires a reliable measurement of task similarity. The measurement of similarity of multiple MDP is an actively studied topic (Ammar et al., 2014; Wang et al., 2019a; Visús et al., 2021), with no single clear solution yet. Additionally, it is not certain that the most similar task will also lead to the best transfer performance. Instead of relying on heuristic similarity measurements, we instead propose to use a data-driven approach and directly predict transfer performance from language instructions.

When confronted with a new task, described using language (the transfer instruction), the best base policy is selected, and the new task is learned more efficiently, based on the parameters of this base policy. A summary of the proposed method can be found in Algorithm 1.

Algorithm 1: Summary of the proposed task-adaptation method

```
α: k instructions sampled from the set of possible instructions Z
β: p instructions sampled from the set of possible instructions Z
foreach instruction i ∈ α do
    Train base policy π_i until convergence
    foreach instruction j ∈ β do
        Sample task-adaptation π_i^j during n training steps. A new policy
          π_j is developed starting with the model parameters from base
          policy π_i.
Train the transfer model
```

5.3.1 Pre-training Base Control Policies

In the pre-training phase, k base control policies $\{\pi_0, ..., \pi_k\}$ are trained. A single base control policy $\pi_i(s_t)$ determines the action $a_t \in \mathcal{A}$ an agent takes, based on the state $s_t \in S$ the agent resides in. Each base control policy should reliably be able to perform one instruction i. This task instruction is expressed in language (e.g., *go to the blue ball* or *pickup the yellow key*).

Training base control policies can be done using any RL algorithm, capable of learning and forgetting new tasks. The amount of pre-trained control policies should be sufficiently large, but smaller than the entire set of possible instructions ($k \ll |Z|$).

The proposed method can be used with a fixed number of base control policies, which are trained during a single pre-training phase. Additionally, our method can also be extended to work in an iterative fashion. In this iterative approach, the agent starts with a small set of k pre-trained base control policies. When confronted with a new task, our method determines the best base control policy to facilitate task adaptation (e.g., π_i). After training the new policy π_j using the model parameters from the selected base control policy π_i, the resulting policy π_j can be added to the set of base control policies. This will allow the execution of more efficient task adaptations, as more base control policies become available.

k instructions $\{z_0, ..., z_k\}$ are selected in order to train base policies $\{\pi_0, ..., \pi_k\}$, from a uniform random distribution. However, an interesting extension to this research might be to select base control policies based on a more advanced selection criterion, such as maximizing the distance between the task instructions in a prior language embedding.

5.3.2 Sampling Task-adaptations

The second phase of the proposed method consists of utilizing the developed base control policies, in order to sample a limited number of task adaptations. A single task-adaptation sample π_i^j consists of taking a fully developed base control policy π_i and using it to perform a new instruction j, different from the one it was trained on. An example of such a sample would include starting from a policy trained on an instruction *go to the yellow box*, and utilize it to perform a different task, such as *pick up the yellow box*.

A task adaptation sample is performed by loading the parameters of the base policy as the initial parameters of the new policy. Training can be performed using any RL algorithm. During this sampling phase the policy does not need to converge. Training only needs to happen for a limited number of n steps. This amount

Instruction z_x	**Transfer instruction z_i**	**Transfer instruction z_j**	**Class**
Go to the green key	Pickup the red ball	Go to the yellow box	1
Go to the red ball	Pickup the red ball	Go to the yellow box	0

Tabel 5.1: Example input dataset, used to train the transfer-model

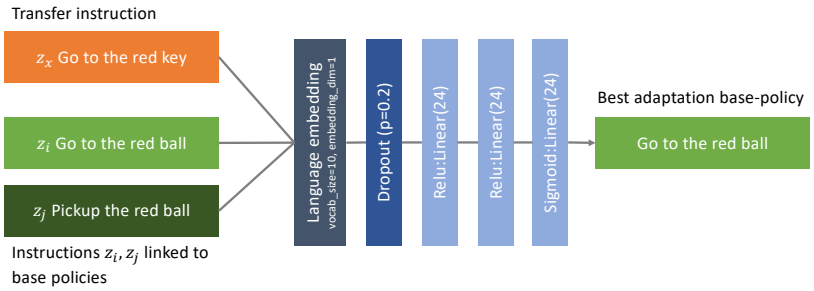
of required steps is significantly lower than fully developing the policy. After the sampled task-adaptation has been executed for n steps, we measure the performance. This can be done by, for example, calculating the success rate of the agent satisfying the instruction over the last 100 iterations. For each base control policy, we randomly select p different tasks from Z to sample task-adaptation.

In summary, in this phase, to develop the dataset, we run $k \times p$ task-adaptation samples (for n training steps). The resulting policies, which are only partially developed, can be used again in a later phase.

5.3.3 Training the Transfer-model

The final stage of the proposed method consists of training a binary classification model: $f(z_x, z_i, z_j) \rightarrow \{1, 0\}$. This model is capable of generalizing the perceived task-adaptation over unseen adaptations, solely from using the task descriptions, formulated in natural language.

The input of the proposed model (Figure 5.2) consists of a concatenation of the sampled transfer instruction z_x, combined with the instructions attached to two candidate base policies (z_i and z_j). The output of the model consists of a single binary output. This output is trained to be 1, if the first base policy with instruction z_i will adapt better than the second base policy satisfying instruction z_j. An example dataset is presented in the Table 5.1.



Figuur 5.2: **Transfer-model architecture**
Two instructions linked to prior developed policies are compared in order to predict the transfer performance on a novel task.

In order to work directly with instructions in language, a language embedding is used. This embedding is trained end-to-end, and thus is specifically trained to encode instructions based on their transfer capabilities.

5.3.4 Transfer-model Usage

The resulting transfer-model can be used when the agent is confronted with a new task description (in language), it currently has no developed policy for. Given a set of labeled base policies, and the new task description, the various possibilities can be tested in order to make an assessment of which base policy will result in the fastest task-adaptation.

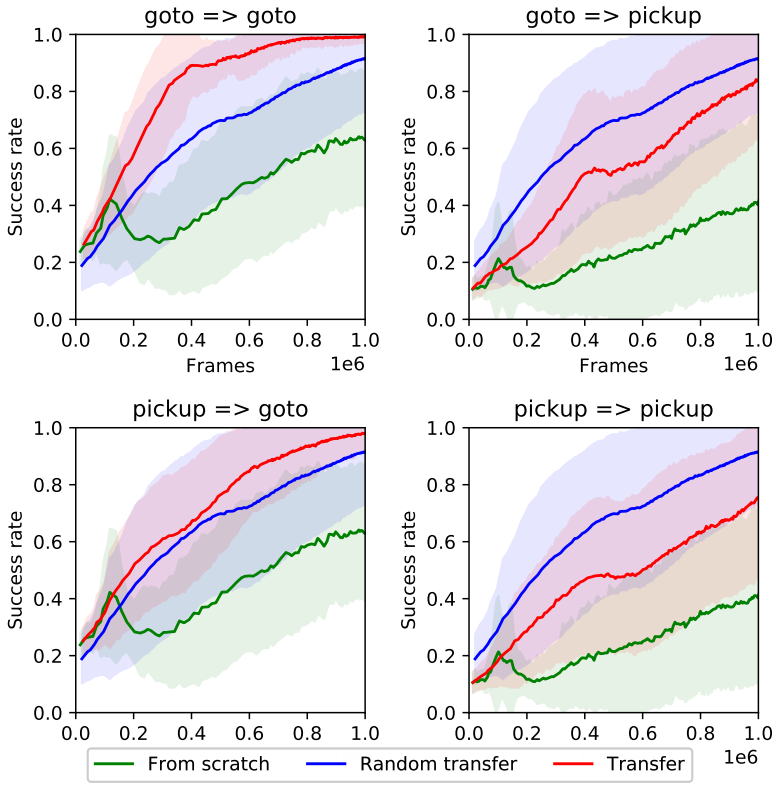
5.4 Empirical Evaluation

In order to find out whether patterns can be discovered in task-adaptations using instructions expressed using natural language, we initially performed a large set of 636 transfer experiments in the BabyAI environment. In each experiment we initialized the new policy with the parameters of a prior fully developed policy, and observed the transfer performance on a new task.

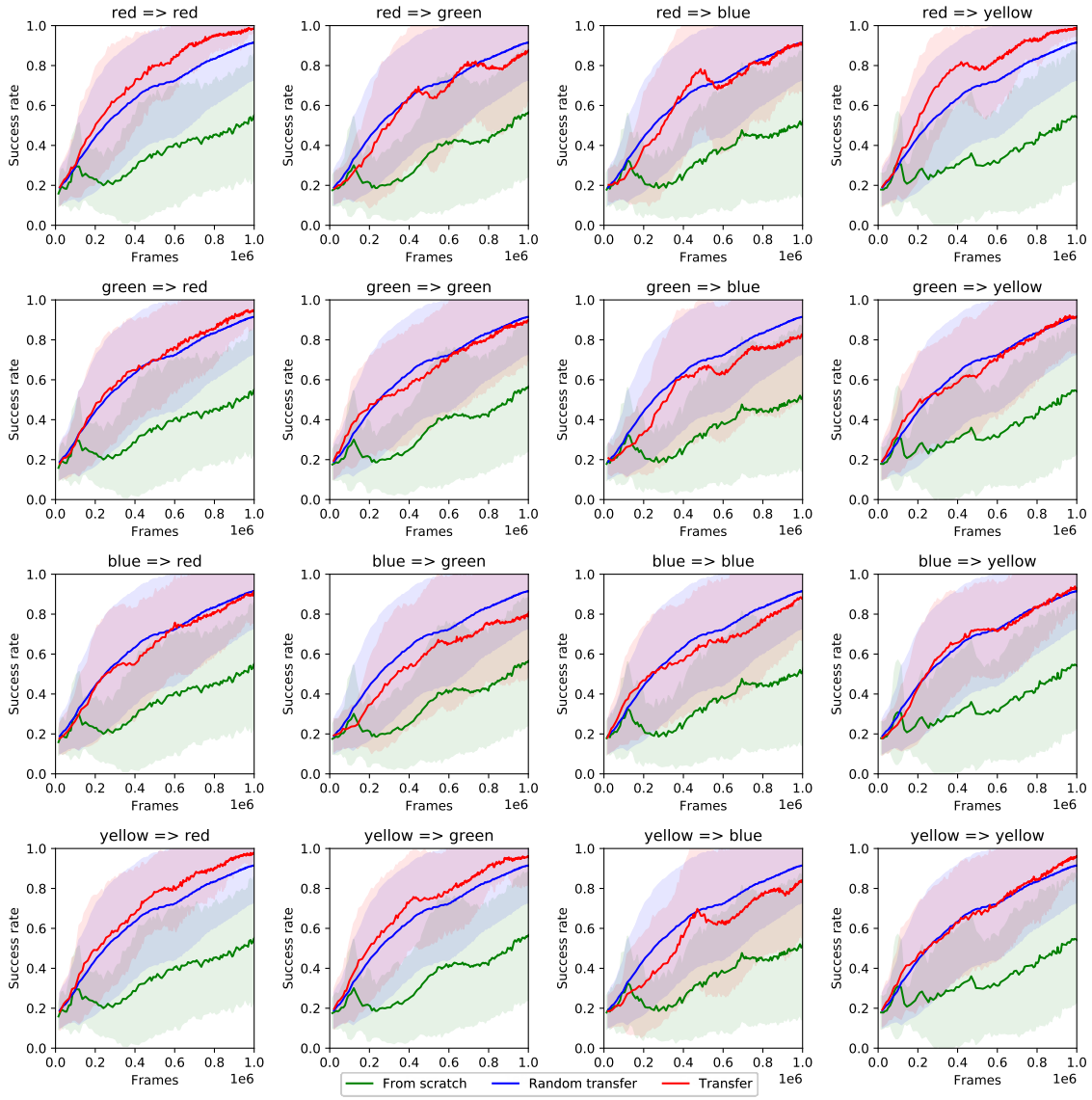
This initial study taught us that complex relations between parts of instructions exist. For example in Figure 5.3 the importance of the verb in the task description is presented. When confronted with a novel *goto* task it is clearly best to also start from a base task trained on a *goto* task. While for a novel *pickup* task, it seems that also starting from a *goto* task is slightly the better option. Similarly, Figure 5.4 shows the effect of the color, and Figure 5.5 the effect of the objects.

However, the discovered relationships between task instructions and transfer capabilities are not straightforward. In some cases the initialization had a positive effect on the transfer performance, however, in some other studied cases the selected pattern (e.g., start from the same color), performed worse than the average baseline (take a random prior pre-trained policy).

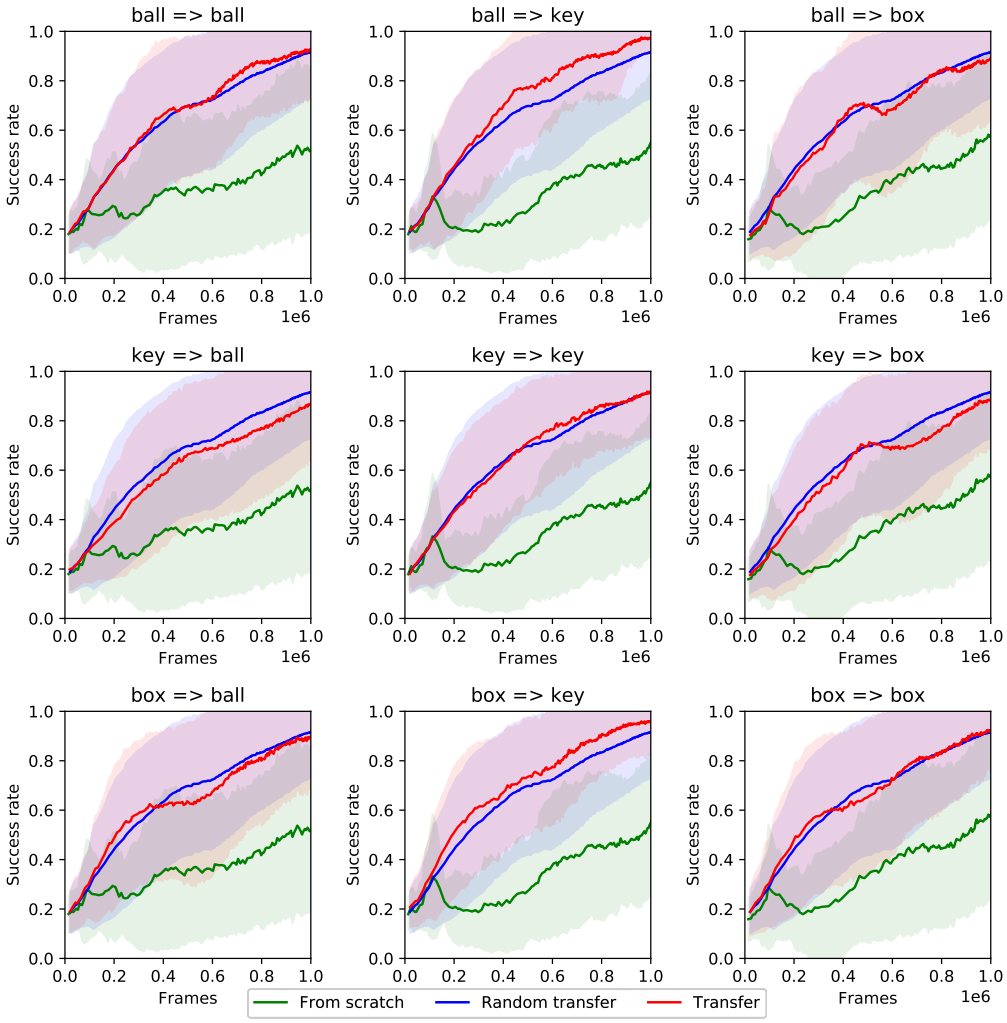
In the second set of experiments, we examined if our proposed classification model could uncover the adaptation patterns, and successfully generalize over the sampled task-adaptations. To perform this, we trained different amounts k of randomly sampled base control policies. While training can be done using any RL algorithm, we used DQN (Mnih et al., 2015) in our experiments. Training a base control policy is done using at least 1M steps, and ends when the policy achieves a success rate of at least 95%, measured on the previous 100 episodes.
After developing k different base control policies, for each base control policy



Figuur 5.3: **Task adaptation with similar/different task verbs**
Comparison of how well different base control policies adapt to new tasks, based on whether the verb in the instruction is the same or different. The solid lines represent the mean, the shading represents the standard deviation. Measured over 636 different transfer tasks.



Figuur 5.4: **Task adaptation with similar/different goal object colors**
Comparison of how well different base control policies adapt to new tasks, based on whether the color in the instruction is the same or different. The solid lines represent the mean, the shading represents the standard deviation. Measured over 636 different transfer tasks.



Figuur 5.5: **Task adaptation with similar/different goal objects**
Comparison of how well different base control policies adapt to new tasks, based on whether the object in the instruction is the same or different. The solid lines represent the mean, the shading represents the standard deviation. Measured over 636 different transfer tasks.

	p=8	**p=10**	**p=12**	**p=14**	**p=18**	**p=20**
k=8	0.61 ±0.03	0.62 ±0.03	0.61 ±0.05	0.64 ±0.05	0.65 ±0.02	0.66 ±0.03
k=10	0.62 ±0.03	0.62 ±0.05	0.64 ±0.06	0.62 ±0.04	0.66 ±0.03	0.67 ±0.02
k=12	0.67 ±0.02	0.67 ±0.01	0.66 ±0.02	0.67 ±0.02	0.68 ±0.02	0.66 ±0.04
k=14	0.64 ±0.04	0.66 ±0.02	0.67 ±0.03	0.69 ±0.01	0.69 ±0.03	0.68 ±0.01
k=18	0.67 ±0.03	0.68 ±0.02	0.68 ±0.03	0.71 ±0.01	0.70 ±0.02	0.71 ±0.02
k=20	0.69 ±0.01	0.68 ±0.05	0.70 ±0.02	0.69 ±0.04	0.71 ±0.03	0.71 ±0.03

Tabel 5.2: Accuracy of the task-adaptation classifier model. The different rows represent the various amount of base control policies used during training (k), the columns represent the number of task-adaptations (p) sampled for each base control policy. Results are averaged over 5 runs.

(n=100,000) p adaptations are sampled. The results gathered from these task adaptations were used to train the transfer model (Adam lr=0.001, 1M steps).

In Table 5.2, we present the performance of our model, using various numbers of base control policies (k), and different numbers of task-adaptation samples (p). We measure model accuracy over a holdout-set consisting of all possible expressible task-adaptations not seen during sampling. This accuracy measures the percentage that our model selected the best base policy (2M steps).

Our results show that even with a limited number of k base control policies, and p sampled task-adaptations, a transfer model can be developed. There is still room for improvement regarding the accuracy of the model, however, the stochastic nature of RL, makes task-transfer inherently noisy.

Nevertheless, efficient task-adaptation realized by the introduced method proves to be a quintessential building block in lifelong learning settings (Silver et al., 2013).

5.5 Conclusion

In this chapter, we presented a method capable of predicting, given a set of base control policies, labeled using language, which of these base control policies will adapt the fastest to a new previously unseen task. In order to make assessments about task-adaptation, our method uses a for this task specifically trained language embedding as part of an end-to-end binary classification model.

The results experimentally demonstrate that this approach is capable of assessing adaptation performance, solely from task descriptions. When confronted with an expanding set of tasks in a lifelong-learning setting, the proposed method has the potential to vastly improve sample efficiency.

Similarly, in a hierarchical setup in which abstractions are utilized, it makes a lot of sense to not start the training process of each abstraction from scratch, but allow the agent to expand its set of capabilities by adapting what it already knows.

6

Task-Adaptation Through Pre-Trained Word Embeddings

The contributions presented in this chapter are based on the publication titled *"Pre-trained Word Embeddings for Goal-conditional Transfer Learning in Reinforcement Learning"*.

6.1 Introduction

In order to build complex intelligent systems, an agent needs to be capable of re-using and adapting previously learned traits. This property is often called the *learning-to-learn* (Lake et al., 2017) ability of an agent.

A *learning-to-learn* approach could allow the agent to become more sample efficient, by allowing the agent to build upon what it already learned in past similar tasks. However, how to implement a *learning-to-learn* system in RL has remained mostly an open question, and is one of the subjects of this thesis. In supervised machine learning with neural networks, training performance on vision tasks can be significantly increased by re-using the initial layers of a previously trained neural network. These initial layers learn to recognize features that are mostly task-independent (Yosinski et al., 2014). Layers on top of these features learn to map combinations of the resulting features to the output labels.

Similar approaches have been used in RL (Taylor and Stone, 2009). Especially in *deep* RL, when working with high-dimensional inputs, it makes a lot of sense to re-use parts of the (learned) visual pipeline across different tasks (Chaplot et al., 2016). In the previous chapter, we proposed a method to make transfer decisions based on the instructions of prior policies and novel tasks.

However, mapping a high-dimensional input to a latent representation is only part of the RL problem. In RL, the agent also needs to explore the environment in order to map actions to states. Such an action can consist of performing a single primitive action, such as *take one step forward*. However, in Chapter 4 we demonstrated that exploration on a higher level of abstraction allows RL approaches to solve more complex problems. These abstractions utilize multiple primitive actions when exploring the environment (e.g., *walk to the garden*).

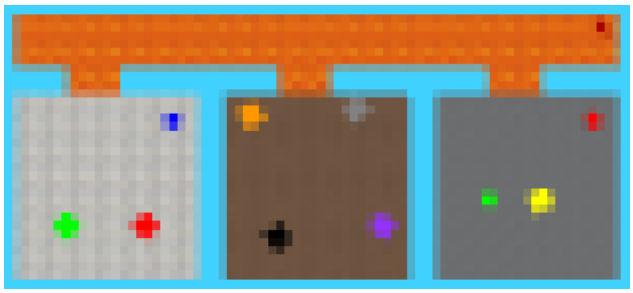
In this chapter, we demonstrate that prior knowledge of a deep RL agent can be used as temporal abstractions in order to facilitate transfer learning to a novel previously unseen task. We do this by utilizing a goal-conditional agent. In this style, the RL agent receives a combination of the current state and a goal vector as its input. This is different from the previous chapter in which a policy was developed for each different task.

Assuming a finite set of possible goals, the goal in a goal-conditional setting is typically represented using a *one-hot* encoded vector. In this one-hot goal space, the distance has no meaning, as the distance between different goals is always the same.

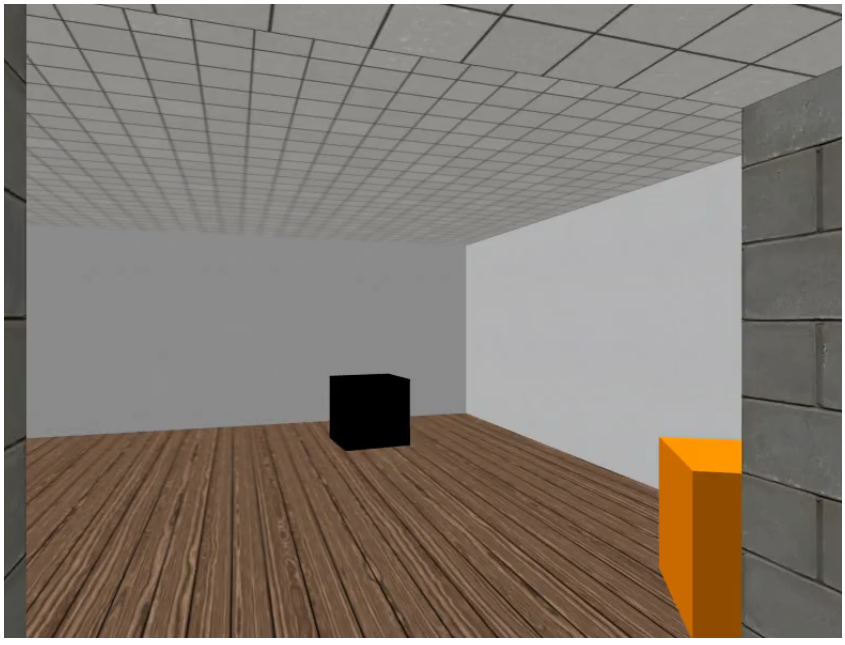
In this chapter, we continue to express the goal of the agent using language. However, instead of learning an embedding from scratch such as we did in Chapters 4 and 5, we now transform words into numbers by utilizing a task-independent pre-trained word embedding. This allows the agent to quickly link a new, previously unseen goal to what it has already learned from past tasks. We experimentally demonstrate that these kinds of pre-trained word goal embeddings can be used to transfer knowledge in the form of temporal abstractions in a transfer learning setting.

6.2 Object Navigation Task Setting

In this chapter, we are concerned with the problem of object navigation (Section 2.6.2.3). In a single instance of this problem, the agent is randomly spawned in a corridor and needs to navigate towards an up-front specified object in the environment. The episode is considered successful if the agent has positioned itself near the goal object in a maximum of 500 steps.



Figuur 6.1: **Top-down layout of the environment used in the experiments**
The three rooms (bathroom, kitchen, bedroom) are connected through a long corridor.



Figuur 6.2: **Egocentric RGB observation MiniWorld**
An example rendering of the viewpoint the agent receives as part of its state.

In order to solve this problem, the agent does not have access to a map of the environment and only needs to rely on RGB sensory input.

For our experiments, we use a custom-designed level in the *MiniWorld* (Chevalier-Boisvert, 2018) environment. This is a similar environment as used in Chapter 4. Figure 6.1 shows the layout of the used environment. Figure 6.2 renders an example observation of the agent.

The designed level mimics a small domestic apartment. Its layout consists of three rooms connected through a corridor. Each room contains a number of typical objects in fixed positions:

- **Bathroom**: shower, bathtub, toilet
- **Kitchen**: stove, toaster, table, microwave
- **Bedroom**: bed, wardrobe, nightstand

Objects are represented with spheres and cubes in different arbitrarily chosen colors. For example, in Figure 6.2, the black box represents the *table* object. We have chosen this abstract setup in order to be capable of fully focusing on the transfer behavior without the interference of other complex sub-problems.

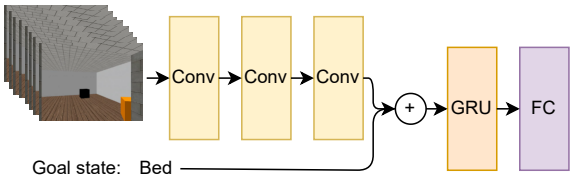
We re-use the reward function introduced in Chapter 4. After taking an action, the agent receives a reward that is equal to the improvement of the distance to the goal object. A *slack* penalty of -0.01 is added to the reward, in order to force the agent to move. A bonus reward of 10 is awarded when reaching the goal object.

6.3 Task-Adaptation Method: Prior Embedding Approach

6.3.1 Goal-encoding

In order for an RL agent to be capable of executing multiple tasks or abstractions, the required task can be specified to the agent using a goal vector (Section 3.7). In our problem setting, this goal vector should correspond with the object the agent needs to navigate to.

Typically, in order to encode different goals, a discrete *one-hot* encoding is used. Unfortunately, when utilizing such a vector, the number of goals should be known in advance, as it is not straightforward to alter a neural network that depends on this vector.



Figuur 6.3: Goal-conditional architecture

However, in the lifelong learning setting (Silver et al., 2013) studied in this thesis, we would like the agent to be capable of learning to navigate to new goals, without having to explicitly define the number of goals in advance. In order to support this, we propose encoding goals using a pre-trained word embedding.

Such a model is typically trained (Mikolov et al., 2013) by taking as input a large corpus of texts and outputs a vector space. Words that appear in similar contexts are trained to be also close to each other in the output vector space. We reason that this prior knowledge can be of great use in a multi-task object navigation task and that goals closer in word vector space will also transfer better between different RL policies.

The pre-trained model we use (Honnibal and Montani, 2017) is trained on the OntoNotes 5 (Weischedel et al., 2013) dataset. This dataset contains a large set of different types of documents and is not linked in any way with our task setting. The resulting model is capable of expressing a goal description with continuous vectors of size $\mathbb{R}^{300}$.

6.3.2 Training Architecture

In order to allow our agent to solve object navigation tasks, we use a standard DRQN architecture (Hausknecht and Stone, 2015). We use the *recurrent* flavor (with sequence-length 8) of the DQN algorithm (Mnih et al., 2015) because the current state does not contain enough information for the agent to successfully navigate the environment. The goal vector is concatenated with the visual perception part of our architecture. This architecture is displayed in Figure 6.3.

6.3.3 Transfer

In order for our lifelong learning agent to be capable of transferring knowledge from one task to another task, we utilized the parameters of a prior policy in order to bootstrap the new policy in Chapter 5.

Within this chapter, we propose an alternative, which consists of adapting the ϵ-greedy exploration scheme (Watkins, 1989). In this scheme, the agent takes a random action ϵ percent of the time, instead of greedily following the current policy π. This allows the agent to explore (potentially better) actions, it would normally not take under the current policy. This ϵ value is typically decayed during training as the agent becomes more confident in its policy. Taking into account experiences collected through following a different policy (e.g., a random or prior policy) requires an off-policy learning algorithm.

We propose to instead of purely taking random exploratory actions in order to navigate to a new goal (e.g., *bathtub*), to also explore actions that would correspond to the action the agent would take if it would be provided with a different goal-vector which the agent already has mastered before (e.g., *shower*).

However, how can the agent know which goal vector will transfer best to satisfy the new unseen goal? We propose to solve this question by measuring the cosine similarity of the unseen goal object and the mastered goal objects in their word embedding space. As these embeddings are trained to put words that are often related to each other close to each other in the vector space, we reason that goals close in this space will most commonly also be located in similar positions in typical building layouts.

Intuitively using knowledge from a prior object goal allows the agent to use this knowledge as a form of temporal abstraction, which corresponds to navigating to the room the object can most likely be found.

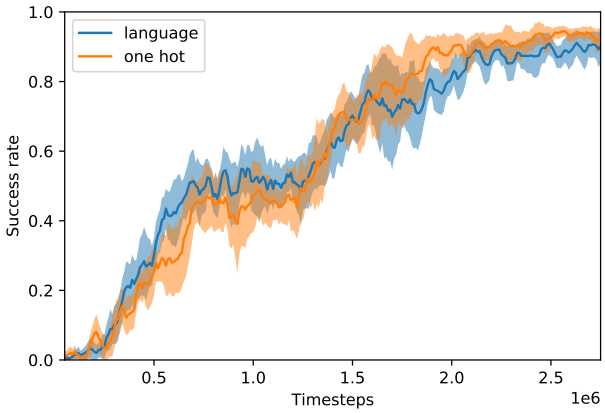
It however remains essential that the agent keeps doing enough exploration, especially in states close to the prior goal object. We propose to introduce a sampling rate hyperparameter α in order to balance the trade-off between biased sampling from the prior policy, and random exploration.

In summary, the policy of our agent when tasked with reaching goal z, word embedding $\mathcal{M}$ and prior goals $\omega_{0...i} \in \Omega$ consists of the following parts:

- $P(1-\epsilon)$: take greedy action $\pi(s_t, z)$
- $P(\epsilon * \alpha)$: sample action from $\pi(s, \omega)$ with $w = argmax_w(cos(M(z), M(w))$
- $P(\epsilon * (1-\alpha))$: take random action

6.4 Empirical Evaluation

Experiments are terminated after reaching a success rate of 0.95 on the last 100,000 steps (and only minimal exploration $\epsilon = 0.01$ is done). In all experiments, ϵ is



Figuur 6.4: **Comparing one-hot encoding vs language goal-vector**
Experiment completed on a set of 4 goals. Results are averaged over 3 runs.

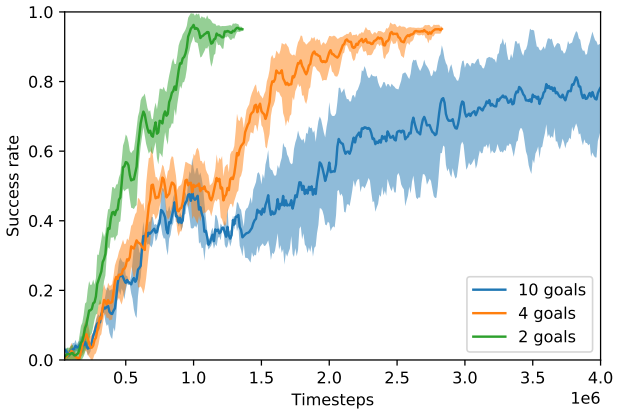
linearly decayed over 1M steps, and we use an experience replay buffer of size 500,000.

6.4.1 Using Language Goal-vector vs One-hot Goal vector

In our first experiment, we examine the impact of the goal vector on the training performance when training a goal-conditional agent on a set of four different goals.

The results of our experiments presented in Figure 6.4 give an indication that directly specifying the goal object using the word embedding ($\mathbb{R}^{300}$) has no significant negative effect over using a one-hot goal object encoding ($\mathbb{R}^{10}$). There also seems to be an interesting relation that using the word goal descriptions has a slightly positive effect on exploration, and using the one-hot encoding seems to work better when the policy is almost ($\epsilon = 0.01$) completely greedy (after 1M timesteps).

Using the goal word embedding for our lifelong learning agent is ideal, as we do not need to specify the amount of possible goal objects upfront. The word goal embedding allows us to input a large number of goals (the used model has 20k unique vectors).



Figuur 6.5: **Comparing training performance on different sizes of goal object sets** Results are averaged over 3 runs.

6.4.2 Initial Training on Limited Goal Sets

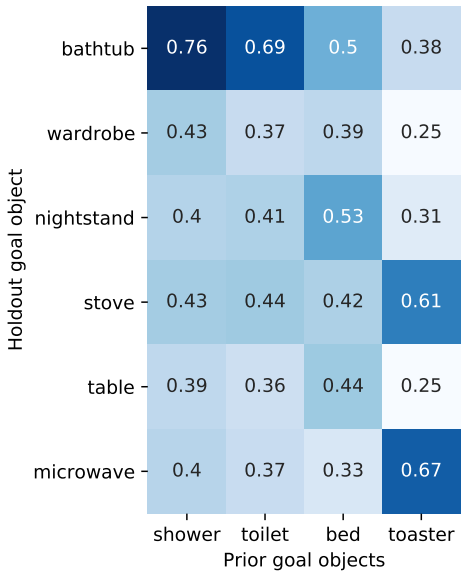
We would like our lifelong agent to be capable of navigating to as many goals as possible. In order to do so, we could train our agent on a large set of goals. However, research has demonstrated (Narvekar et al., 2020) that using a carefully selected task curriculum often leads to better results.

We plotted the results of training our agent using different sizes of goal sets, in Figure 6.5. These results demonstrate that initial larger sets of goals are significantly harder to train. This finding supports our claim that a lifelong learning agent significantly benefits from first learning a small sub-set of goals, and gradually expanding its capabilities through transfer learning.

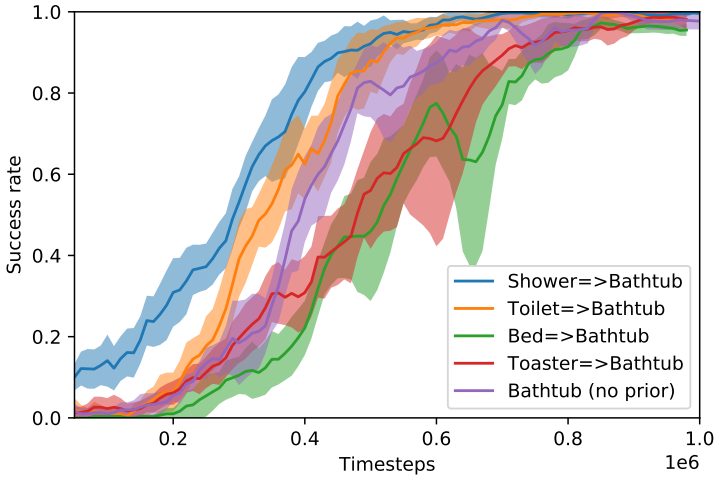
6.4.3 Transfer to New Objects Using a Prior Policy

In the final experiment, we allow the agent to transfer knowledge from one goal object to a different unseen goal object using the transfer mechanism described in Section 6.3.3.

We start with a policy that has been trained to reliably reach four goal objects in the environment (*shower, toilet, bed* and *toaster*). In this experiment, we test the transfer capability of our algorithm in order to learn to reach a new goal object *bathtub* using a prior sampling rate of $\alpha = 0.2$. The new policy is randomly initialized.



Figuur 6.6: **Cosine similarity of holdout goal objects and prior goal objects in the word embedding**



Figuur 6.7: **Comparison of using different prior goals in order to learn how to reach a new unseen goal object (*bathtub*)**
Results are averaged over 3 runs.

The results of this experiment, plotted in Figure 6.7, demonstrate that the goal object that transfers best to the new unseen goal object (*bathtub*) is *shower*, which is also the goal object that is closest in language space (Figure 6.6). The performance when using the second-closest goal (*toilet*) in language space also performs similarly.

Unrelated goals such as *bed* and *toaster* hinder the agent, steering the agent to the wrong room (*kitchen*) and we observe a negative transfer effect compared to just learning to navigate to the goal without any prior knowledge.

6.5 Conclusion

In this chapter, we presented our preliminary ideas on how language can assist an RL agent in a lifelong learning setting through the usage of prior knowledge contained in word embeddings. The work done in this chapter removes the dependency on expensive adaptation sampling (Chapter 5), and serves as the foundation for the *Disagreement Options* method developed in the next chapter.

The proposed approach consists of training the agent on small sets of goals, directly inputting the goal descriptions in natural language. We utilize the similarity of descriptions of seen and unseen goal objects in language in order to decide how to transfer existing knowledge to novel tasks. In order to transfer knowledge, we propose a simple, but effective transfer mechanism. We support the method with results in a 3D simulated domestic environment.

7

Disagreement Options

The contributions presented in this chapter are based on the publication titled *"Disagreement Options: Task Adaptation Through Temporally Extended Actions"*.

7.1 Introduction

Humans acquire a wide range of different skills over a lifetime. We are capable of solving complex new problems by quickly adapting, and combining these skills. For example, when learning how to ride a motorbike, balancing skills learned from riding a bicycle might be re-utilized.

But how do we know which prior skills can be useful when confronted with a new task? We could use *trial-and-error* learning, and test which of our prior skills works best in a new situation. This approach is commonly used in HRL approaches (Chapter 3).

However, when able to communicate, language is a much more efficient instrument to communicate how different skills can be transferred, in order to solve new tasks (Chapters 5 and 6). For example, one could say to someone who is learning how to ride a motorbike that: *riding a motorbike is just like riding a bicycle*. Or, in order to find a new object, one typically can explain how to find it in terms of the

relation with other objects we already are able to localize: e.g., *the microwave is on top of the fridge.*

As discussed before, current RL approaches typically start training from scratch and offer no solution on how to efficiently extend the capabilities of an agent over its lifetime (Silver et al., 2013). This is especially an important problem in real-world embodied systems (e.g., a collaborative robot). In this setting, an agent typically has no access to large amounts of computing resources and needs to come up with new solutions in a reasonable timeframe.

In order to work towards real-world embodied systems, capable of quickly adapting their knowledge to novel tasks, inspired by the way humans learn through communication, we introduce a novel HRL method in this chapter. Our overarching method simultaneously formulates an answer to two important questions:

- Which prior skills are useful when learning how to solve a new task? (also studied in Chapters 5 and 6)
- How can we solve the trade-off between utilizing prior knowledge, and acquiring new skills by exploring the environment? (also studied in Chapters 4 and 6)

We answer these questions by utilizing pre-trained word embeddings introduced in Chapter 6 to select source tasks based on their goal descriptions in natural language. We utilize the disagreement between prior policy action distributions in order to decide when to exploit the priors, and when to explore novel paths.

Our answers to these two questions allow an agent to use prior knowledge efficiently as temporally extended actions.

7.2 Policy Training

In order to train a policy, we opted to use Sample Efficient Actor-Critic with Experience Replay (ACER) (Wang et al., 2017b) as it utilizes recent variance reduction techniques, parallel training, and off-policy updates using an experience replay buffer. More specifically we choose ACER because of the following properties:

- Focus on sample efficiency through the usage of an experience replay buffer, which allows the usage of environment experiences multiple times.
- Off-policy updates through importance sampling allows for our adaptation method to utilize actions sampled from a different distribution (the prior policies).

- The policy directly outputs a distribution over actions which we can compare with other policies.

In ACER on each training iteration, there is an on-policy update after taking n rollout steps. Afterward, there are also one or multiple off-policy updates by taking samples from a replay buffer.

7.3 Method

Our method is concerned with utilizing prior knowledge as temporally extended actions (options) in order to increase the sample efficiency, the required interactions with the environment, when learning new tasks.

The approach can be divided into two distinct sub-systems, which each address an important question. The *task similarity* system (Section 7.3.1) is concerned with selecting useful prior knowledge which will be best suited in order to solve the novel task. For example: would a *bicycle riding* skill be more useful than a *car driving* skill when learning how to ride a motorbike? Once we have selected which priors we would like to use, the *task adaptation* phase (Section 7.3.2) is initiated in order to train a new policy by intelligently reasoning when to utilize prior knowledge as temporally extended actions, and when to explore the environment. The agent assumes the presence of a set of prior policies, we discuss some possibilities on how to acquire such priors in Section 7.3.3.

The pseudocode of the entire approach is presented in Algorithm 2.

7.3.1 Task Similarity: How to select relevant priors?

The agent is provided with a set of different prior policies $\{\pi_{g_1}, ..., \pi_{g_i}\}$, all capable of reliably performing one or multiple different tasks $\{g_1, ..., g_i\}$. In order to decide which prior policies are useful as prior knowledge when learning a new task, we make use of natural language. The reasoning which we also introduced in the previous chapter, is that when goal descriptions are close in language space, they are potentially also close in policy space (Fulda et al., 2017).

More specifically, we use the same pre-trained word embedding from the previous chapter (Honnibal et al., 2020). This embedding was pre-trained on a set of tasks that are not tailored to our setting, utilizing the *OntoNotes 5* (Weischedel et al., 2013) dataset. Our embedding is trained (Mikolov et al., 2013) by taking as input a large corpus of texts and outputs a vector space $\mathbb{R}^{300}$. Words that appear in

Algorithm 2: Disagreement Options

$\mathcal{M}(\cdot)$: Pre-trained word-embedding
$\mathcal{B}$: disagreement score buffer with max size α
$\pi(s_t, g_t)$: new policy under training

while *agent rollout in progress* **do**
  Observe state s_t and goal g_t
  $x \sim \mathcal{U}(0, 1)$
  if $x < \mathcal{H}(\pi(s_t, g_t)) - 0.1$ **then**
    Find 2 closest prior policies (π_{z1}, π_{z2}) according to:
    $z_i = argmax_{z_i}(cos(\mathcal{M}(g_t), \mathcal{M}(z_i))$

    Calculate disagreement score:
    $d1 = D_{KL}(\pi_{z_1}(s_t, z_1)||\pi_{z_2}(s_t, z_2))$
    $d2 = D_{KL}(\pi_{z_2}(s_t, z_2)||\pi_{z_1}(s_t, z_1))$
    $d = min(d1, d2)$

    Add disagreement score to buffer $\mathcal{B}$

    if $\sum_i^{\alpha} \mathcal{B}_i/\alpha > \beta$ **then**
      $x \sim \mathcal{U}(0, 1)$
      if $x < 0.5$ **then**
        Perform action $a_t \sim \pi_{z_1}(s_t, z_1)$
      else
        Perform action $a_t \sim \pi_{z_2}(s_t, z_2)$
  else
    Perform action $a_t \sim \pi(s_t, g_t)$
  Store $\langle s_t, a_t, s_{t+1}, g_t, r_{t+1} \rangle$ in ACER experience replay buffer
Perform ACER on-policy update
Perform n ACER off-policy updates

similar contexts are trained to also be close to each other in the resulting vector space.

When confronted with a new goal g_t, we calculate the cosine similarity of the resulting vector, after being processed through the word-embedding $\mathcal{M}(x)$ with all labels $\{z_0, ..., z_i\}$ attached to the available prior policies $\{\pi_{z_0}, ..., \pi_{z_i}\}$:

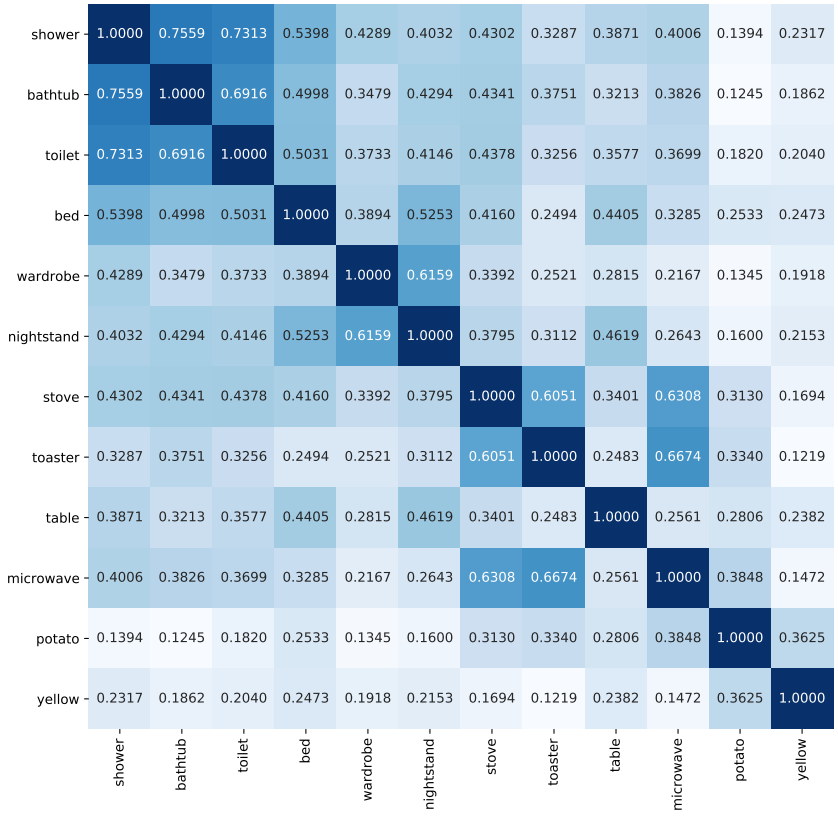
$$z_i = argmax_{z_i}(cos(\mathcal{M}(g_t), \mathcal{M}(z_i)) \tag{7.1}$$

We select the two policies whose labels are closest to the new goal in the word-embedding space as prior knowledge. Our method requires at least two policies in order to calculate a disagreement between their action distributions in the next phase. We use the minimum of two prior policies in the rest of this chapter, as prior knowledge is often expensive to acquire. However, it's a straightforward extension to adapt our method to use more priors. The cosine similarity between goal objects used in our experiments is pictured in Figure 7.1. For example, in an *ObjectNav* task, when asked to navigate to a new goal object *shower*, policies attached to goals such as *bathtub* and *toilet* are most similar in the word-embedding space, and will be selected (if available) as most potent source tasks.

7.3.2 Task Adaptation: How should we use the prior knowledge?

Once we have selected the prior policies which we expect to be most useful, we can utilize these priors in order to solve the novel task. We treat the selected prior policies as options (Section 3.6). Thus, the agent now needs to decide when to use its primitive actions in order to explore, and when to follow the option policies in order to quickly reach new parts of the state space. This explore/exploit challenge, is one of the key challenges of RL. While this problem is typically addressed on the low-level primitive action level, in this chapter we address this problem on a higher level of abstraction.

This setting requires a delicate balance because when the agent would only follow the temporally extended actions greedily, it would not be capable of learning anything new. So, ideally, the agent should be capable of assessing when it should greedily follow the priors, and when it should explore. For example, when we are trying to locate a *toothbrush* object in a house, a temporally extended action that would take the agent to the bathroom is a useful prior. However, once we have entered the bathroom, the agent should explore within it, in order to extend its capabilities.



Figuur 7.1: **Similarity scores of different goals in the word-embedding space** These scores are used in order to decide what prior knowledge to use.

Note that if the agent had access to a sensor that knows in which room the agent resides, this sensor could be used to steer the termination of the active option. Unfortunately, such a sensor is not commonly available, and we propose an alternative scheme based on the disagreement between priors, to steer option termination.

In order to decide when to use prior knowledge, we utilize the action distributions of the selected prior policies. Given a state s_t these prior policies output different action distributions. We reason that when these distributions align, measured by the Kullback-Leibler (KL) divergence between them, it is useful to greedily follow these policies as a temporally extended action. We call this score the *disagreement score*.

$$D_{KL}(\pi_{z_1}(s_t)||\pi_{z_2}(s_t)) = \sum_a \pi_{z_1}(a|s_t) \log \frac{\pi_{z_1}(a|s_t)}{\pi_{z_2}(a|s_t)} \tag{7.2}$$

Because the KL divergence is not symmetric, we calculate the disagreement score as follows:

$$d = min\left[D_{KL}(\pi_{z_1}(s_t)||\pi_{z_2}(s_t)), D_{KL}(\pi_{z_2}(s_t)||\pi_{z_1}(s_t))\right] \tag{7.3}$$

By using the minimum we slightly favor utilizing the prior knowledge, which experimentally yielded the best results.

When the two prior policies diverge on what the action of the agent should be, we terminate the temporally extended action and let the agent explore by itself. For example, two policies that pursue a *towel* and a *toothbrush* object, will have similar action distributions up until they reach the bathroom. Upon entering the bathroom the action distributions diverge because their implicit high-level navigation target changed from reaching the bathroom to reaching the individual objects.

Because the action distributions of the prior policies can be noisy, we utilize a moving average of the disagreement scores B acquired over the last α steps. On each training step, we compare this moving average against a threshold β in order to decide when to use our prior knowledge, and when to terminate the temporally extended action:

$$a_t = \begin{cases} \pi_{z_1}(s_t), & \text{if } \sum_i^{\alpha} B_i/\alpha > \beta \\ \pi(s_t), & \text{otherwise} \end{cases} \tag{7.4}$$

When the prior policies are in agreement, we randomly sample the recommended best action from one of the prior policies. As their divergence is small, they will output similar actions, so it does not matter which one to sample from. We take this action in the environment and use it to update the new policy. In contrast, if

there is disagreement, the agent uses the new policy to explore, by sampling an action from it.

While the disagreement window α and the disagreement threshold β are hyper-parameters, which potentially are subject to an expensive search in order to get optimal values, we experimentally demonstrate that approximate optimal values can be found easily.

Because ACER has an experience replay buffer, and utilizes off-policy training, after a few iterations, prior knowledge will have found its way into the buffer, and thus also into the new policy. In order to gradually reduce the dependency on the priors, we only rely on the priors when the entropy of the action distribution of the new policy for the current observed state $\mathcal{H}(\pi(s_t, g_t))$ is still high. We assume this distribution entropy lowers as the new policy learns the new task. This is a realistic assumption in a deterministic environment in which an optimal policy will converge to assigning almost all probability to a single action given a state. The entropy measurement is used to gradually reduce the probability of invoking the temporally extended actions:

$$\mathcal{I}(s_t) = P(x \sim \mathcal{U}(0,1)) < \mathcal{H}(\pi(s_t)) - 0.1 \tag{7.5}$$

We correct this probability with a small factor -0.1 in order to encourage exploration early on in training. Increasing this factor will reduce the usage of the priors.

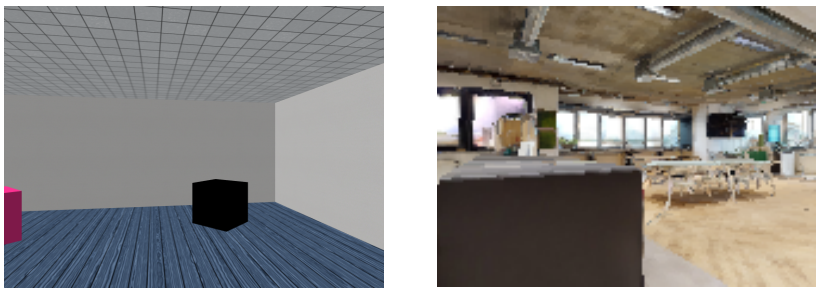
7.3.3 Prior Policy Acquisition

We assume prior policies are provided a priori to the agent. A lot of different options are available to acquire such source policies. One could use any RL algorithm to train a policy. We especially envision RL methods that maximize entropy to be potent methods to acquire diverse prior policies. For example, VIC (Gregor et al., 2016) tries to maximize the number of different states the agent can reach by maximizing the mutual information between the set of skills and their termination states.

We also deem it possible to use an imitation learning approach (Ho and Ermon, 2016; Ross et al., 2011) to bootstrap the agent, utilizing policies compiled from (human) expert demonstrations.

7.4 Empirical Evaluation

This section empirically demonstrates the effectiveness of the introduced method in two different settings: a simple 3D *gridworld* and the photo-realistic Habitat simulator.



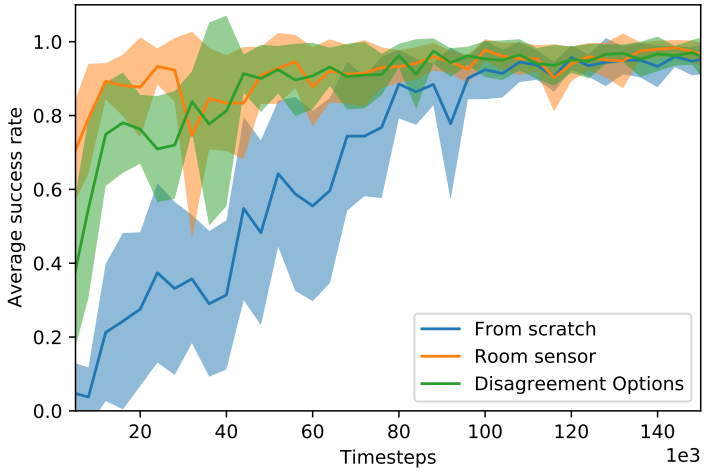
(a) MiniWorld environment (b) Habitat environment

Figuur 7.2: Example egocentric RGB states used in the experiments

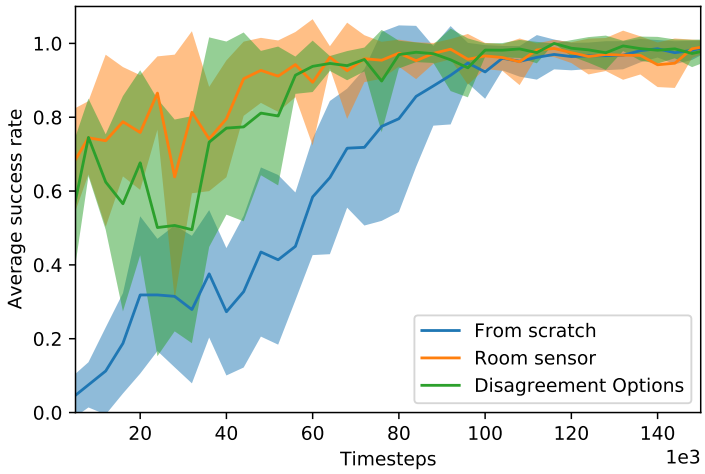
7.4.1 3D MiniWorld

The setting of our first set of experiments consists of a visually basic 3D world also used in the empirical evaluation in Chapter 6. In this environment, we simulate a domestic apartment setting with three fixed different designated rooms: a bedroom, a kitchen, and a bathroom. Each room has a visually distinct theme and has multiple objects in it. The objects are represented using differently colored cubes in fixed positions. These three rooms are connected by a corridor. The agent always starts in a random position in this corridor. This setting is implemented as a custom level in the *MiniWorld* (Chevalier-Boisvert, 2018) environment.

In each episode, the agent is tasked with finding an object in this environment. The state space consists only of the egocentric RGB render (e.g., Figure 7.2a). Additionally, the agent observes a densely defined reward signal, which consists of the decrease of distance between the agent and the goal object. We also penalize the agent for slacking by subtracting a negative reward of -0.01 for each step taken. A positive reward of 10 is rewarded upon reaching a minimum distance to the goal object. The agent is allowed a maximum of 500 steps to reach the goal.

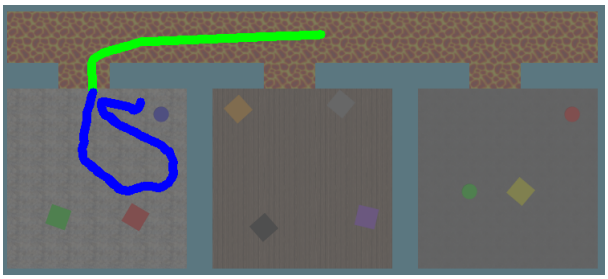


(a) New goal object: **bathtub**, priors: shower, toilet



(b) New goal object: **nightstand**, priors: bed, wardrobe

Figuur 7.3: **Average success rate of our disagreement agent (green) and our disagreement agent with a room sensor (orange)**
We compare with learning the task from scratch (blue). Results are averaged over 10 runs and utilized window size $\alpha = 10$ and disagreement threshold $\beta = 0.1$.



Figuur 7.4: **Example trajectory of the disagreement agent followed during training** In the green part of the trajectory the agent follows the prior, in the blue part the agent explores the environment. In this case, the agent has access to a room sensor and only explores in the room of the goal object.

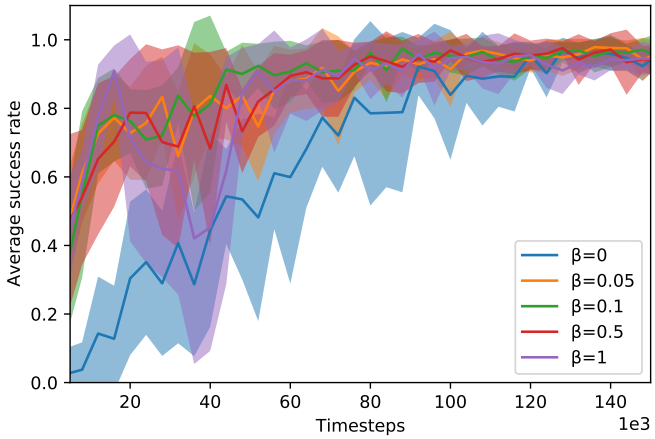
7.4.1.1 Room Sensor

In order to validate our hypothesis that prior knowledge can be useful to navigate the agent to the room with the goal object in it, we first equip the agent with a room sensor. This sensor informs the agent when it is positioned in the corridor, and thus should follow the prior policies greedily, in order to navigate to the room containing the goal object. We selected prior policies which were trained on goal objects that are in the same room as the new goal object. Once inside the correct room, the agent knows not to follow the prior anymore, but to explore by itself.

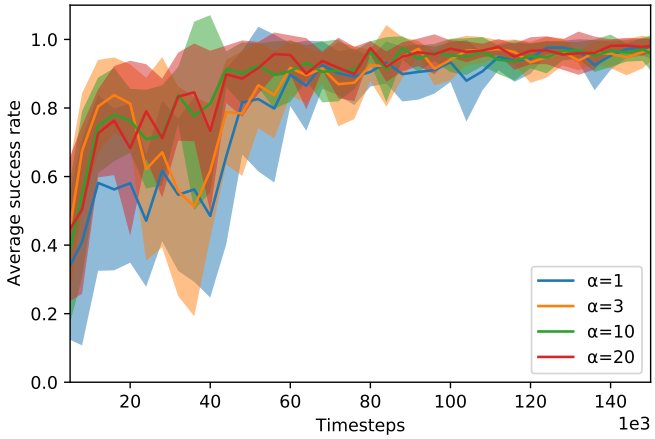
When utilizing this room sensor with prior policies capable of navigating to the *shower* and *toilet* goal objects, our results show that the agent almost instantly (50k training steps) is capable of adapting to reliably reach the new *bathtub* goal (Figure 7.3a). Similarly, the agent is capable of quickly learning to navigate to the *nightstand* goal object using prior policies capable of reaching the *bed* and *wardrobe* (Figure 7.3b). We plot an example trajectory followed during training in Figure 7.4. In this trajectory, the usage of prior knowledge that led the agent to the correct room is plotted in green, while the exploratory part of the trajectory is plotted in blue.

7.4.1.2 Disagreement Options

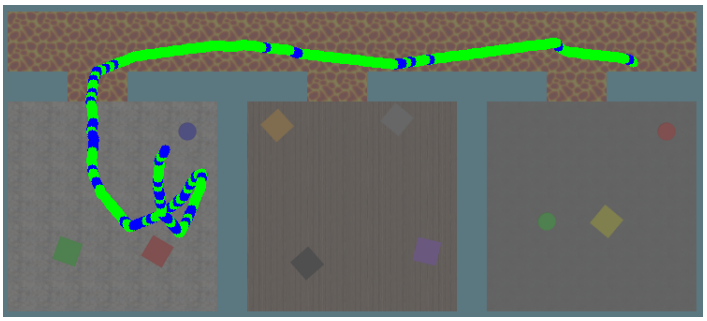
However, a room sensor is not something an autonomous agent typically has access to. In the second set of experiments, we wanted to validate whether the disagreement options provide a similar efficient usage of prior knowledge without such a sensor. As plotted in Figure 7.3, the agent is capable of efficiently utilizing the prior knowledge when using the disagreement scheme ($\alpha = 10$, $\beta = 0.1$), starting with a success rate averaging 60-80%, and quickly getting an average success rate



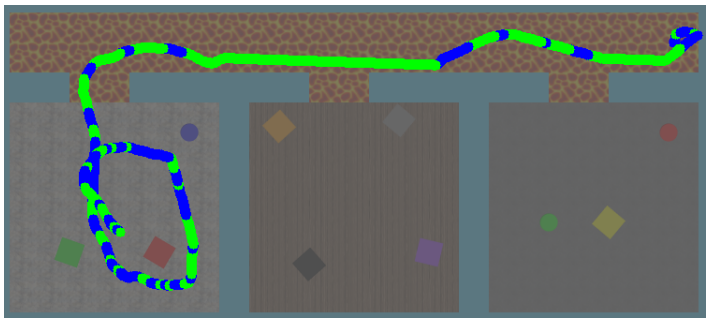
Figuur 7.5: **Ablation study of the disagreement threshold in the *MiniWorld* environment**
A value of 0 never utilizes the prior knowledge, while a value of 1 does not explore the environment (when the action distribution entropy is still high at the beginning of training). Results are averaged over 10 runs. (new goal: bathtub, priors: shower, toilet)



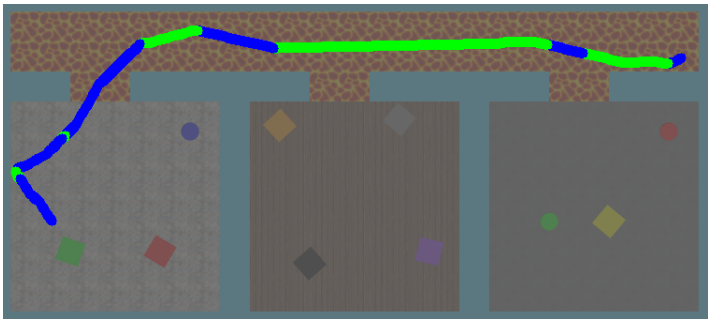
Figuur 7.6: **Average success rate of our disagreement agent in the *MiniWorld* environment on the bathtub task**
We compare different disagreement window sizes. Longer disagreement windows lead to more stable utilization of the temporally extended actions. Results are averaged over 10 runs.



(a) Window size $\alpha = 1$



(b) Window size $\alpha = 3$



(c) Window size $\alpha = 10$

Figuur 7.7: **Example trajectories of the disagreement agent using different disagreement windows in the *MiniWorld* environment**
Parts of the trajectory marked in green utilized the prior knowledge, in blue parts the agent explored. In this setting larger disagreement windows lead to more stable utilization of the prior knowledge.

Figuur 7.8: **An example trajectory of the agent in a scan of The Beacon office building.** The red star is the new goal, while the prior goals are marked with a yellow circle.

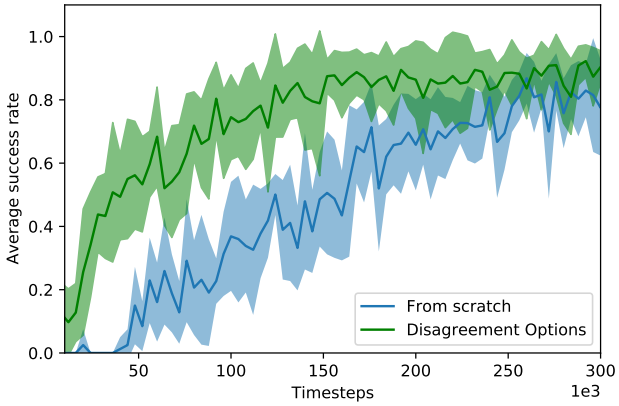
of nearly 100%.

We also did an ablation study of our hyperparameters in this setting. In Figure 7.5, we demonstrate the impact of the disagreement threshold β. When setting the value too high, the agent does not explore enough, while a too low β value will only marginally benefit the task adaptation.

Figure 7.6 presents the impact of the disagreement window size α. In this setting, larger window sizes ($\alpha > 3$) are more efficient, as smaller window sizes lead to noisy trajectories, while a larger window size allows the agent to exploit the prior knowledge more systematically (Figure 7.7).

7.4.2 Photorealistic Simulator

For our second set of experiments, we use the Habitat photorealistic simulator (Savva et al., 2019) and a 3D scan of our office floor (Section 8.2). This environment is considerably more challenging than the *MiniWorld* environment, both structurally and visually. We use the same reward setting as in our *MiniWorld* experiments. Similar to the *MiniWorld* environment, the agent only has access to a



Figuur 7.9: **Results of the disagreement agent in the photorealistic Habitat simulator** Disagreement agent (green) (window size $\alpha = 10$, disagreement threshold $\beta = 0.1$) compared to learning the task from scratch. Results are averaged over 10 runs.

visual RGB egocentric observation of the current state. In this setting, the agent starts in a completely random position and is allowed to take 500 actions in order to reach a new goal in a fixed position (the main table in the office canteen). In order to master this novel task, the agent has access to two prior policies which are capable of navigating to two other goals within the canteen.

The results from using our disagreement options within this environment can be found in Figure 7.9. While this task is considerably harder in terms of structure than the *MiniWorld* tasks, the agent is capable of utilizing the prior knowledge in order to reach a nearly perfect average success rate on the novel task considerably faster (150k training steps vs 250k training steps), than if the agent would have to start from scratch.

7.5 Discussion

Our task-adaptation method is supported by the assumption that goals that are close in language space should also be close in policy space. However, this might not always be the case. If the agent selects prior goals which are physically located nowhere near the new goal, but in different locations, our method will not hinder progress as the priors will always disagree, and thus the agent will not use the priors. If however, the wrong priors do agree on the next action, the agent will be steered in the wrong direction, and learning will be slower. In these settings, the

disagreement threshold could be lowered.

In our experiments, we utilized a deterministic environment. If the environment is completely stochastic (e.g., all objects are randomly placed in random rooms) our method would not be able to utilize prior knowledge. However, if objects are placed in random positions, but always in the same rooms, our adaptation method would still be capable of adapting, and could even benefit from the learned ability of the priors to explore a certain room.

7.6 Conclusion

In this chapter, through building upon the research presented in Chapters 5 and 6, a novel overarching method was introduced to transfer prior knowledge from prior tasks to a new task through temporally extended actions. We do this by selecting prior knowledge based on cosine similarity in a prior word-embedding space. In order to decide when to utilize the selected prior knowledge, and when to explore our environment, we rely on the disagreement between action distributions of the selected priors.

We demonstrate the effectiveness of our method in a visually simple 3D *MiniWorld* and a photorealistic simulator. We also hint at how our method might be used in the real world to expand the capabilities of a real-world embodied agent.

Because our method only relies on goals formulated in natural language and egocentric visual observations, we can also potentially use our method in a real-world setting. In this setting we let the agent solve different tasks in simulation, and through *sim2real* techniques, utilize them in the real world. When confronted with a new task in the real world, the agent could use the prior knowledge gathered in simulation to solve the novel task considerably faster in the real world.

It is often not possible to define a dense reward signal in the real world. The use of prior knowledge allows our agent to efficiently reach states closer to the goal object and thus increases the chance of the agent obtaining positive learning signals. This allows us to believe that it might be possible to learn only from sparse reward signals, which are more obtainable in real-world scenarios. In the final chapters of this thesis, we will lay the foundation for utilizing the developed approaches in such real-world scenarios.

8

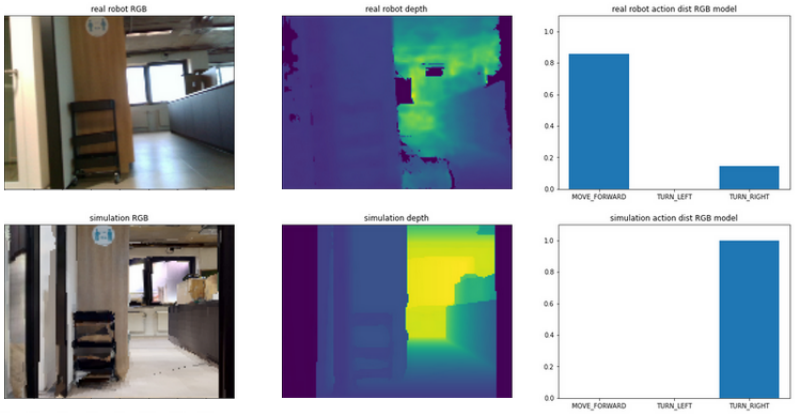
Real-World PointGoal Navigation

The contributions presented in this chapter are based on the publication titled *"A Multimodal AI Approach for Intuitively Instructable Autonomous Systems: A Case Study of an Autonomous Off-Highway Vehicle"*.

8.1 Introduction

The work presented so far focussed on making RL approaches more sample efficient through utilizing abstractions and allowing these abstractions to quickly adapt. Unfortunately, the introduced approaches are still too sample inefficient in order for them to be trained directly in a real-world environment.

In this chapter, we present two alternative approaches in order to use RL approaches in real-world environments. The first approach makes use of a simulator to train the policy. The second approach utilizes a digital twin in order to perform PointGoal navigation in the real world.



Figuur 8.1: **Example RGB and depth observations obtained in the real world (top row), and in simulation (bottom row)**
While the state looks similar to a human observer, possibly due to noise and different lighting conditions, a trained agent takes different actions depending on which type of observations it receives (right bar plots).

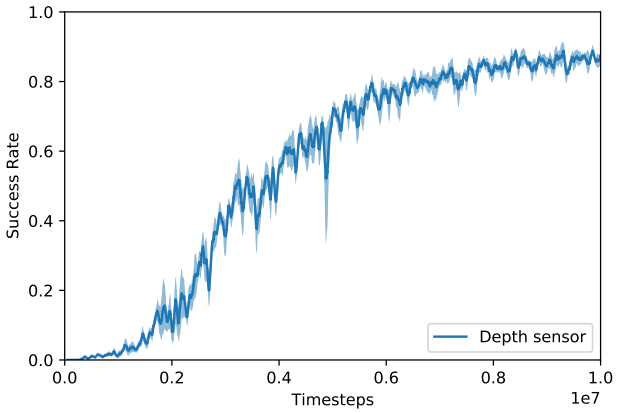
8.2 Sim2Real: The Beacon Office Simulator

In order to train RL in a real-world setting, issues such as safety considerations and automatic resets should be considered (Dulac-Arnold et al., 2019; Zhu et al., 2020).

Instead of directly training on real-world high-dimensional observations a *sim2real* approach has also been researched intensively. In this approach, the RL agent is trained in a simulated environment, and the resulting policy is then directly utilized in a real-world environment.

The sim2real approach is however plagued with two major issues. The first issue consists of the fact that it is difficult to simulate sensors realistically. Real-world sensors are often subject to noise, while their simulated counterparts are not. Similarly, simulated physics is often different from those exhibited in the real world. While most simulators have perfect actuation, real wheels exhibit issues such as *wheel slippage*.

The gap between real-world sensors (Figure 8.1) and their simulated counterparts can somewhat be mitigated by also simulating their noise behaviors. However, a second issue that has often been observed is the overfitting of the learned agent to the simulator (Truong et al., 2022). This is especially a problem if the simulator contains bugs that the agent could exploit. An example of such a discovered bug is *wall sliding* (Kadian et al., 2019). In this case, the simulated agent slightly moves



Figuur 8.2: Baseline training performance of a DD-PPO (Wijmans et al., 2020) agent in the Beacon environment

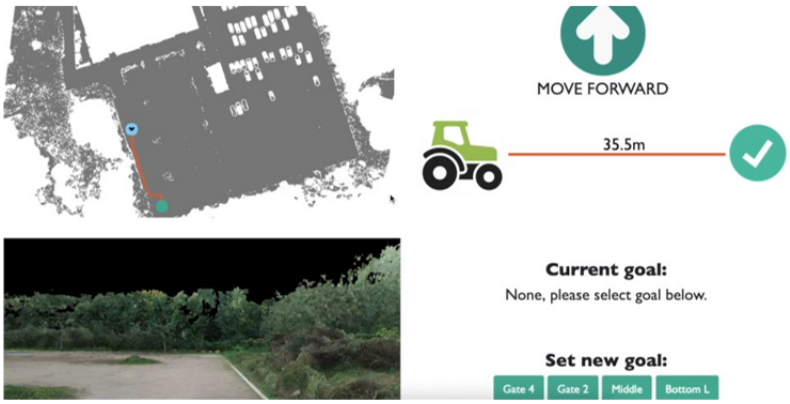
to the left or right when colliding with an obstacle. A navigation policy could then be obtained by deliberately colliding with walls in order to find an opening in a narrow corridor without utilizing any visual sensor information.

In order to study these phenomena a 3D scan of the office building of the IDLab research group (called *The Beacon*) was made. Having easy access to both the real version and the simulated version of the same environment allows researchers and students to work in short feedback loops. This environment has already been successfully utilized in order to complete PointGoal, ObjectGoal, and AreaGoal tasks.

The scan was made using a special *Matterport Pro2* camera. This scan was then compiled in order to become a scene in the Habitat (Savva et al., 2019) simulator.

8.2.1 Training

As a baseline a DD-PPO agent (Wijmans et al., 2020) was trained for 1B steps using 4 workers on the PointGoal task (Section 2.6.2.1). Training this model takes about 26 hours using a Tesla V100 GPU. The performance observed during training is plotted in Figure 8.2



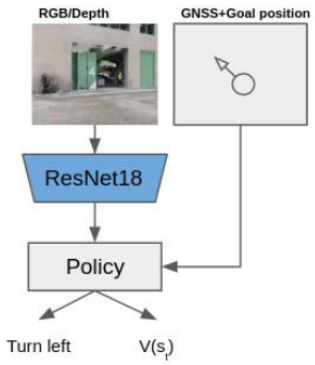
Figuur 8.3: **Digital twin interface**
The planned trajectory is displayed on the top left. The RGB observation of the digital twin is visible on the bottom left.

8.3 Digital Twin

As discussed before in this thesis, RL is too sample inefficient to be directly trained in real-world environments. A common approach is to instead train in simulation, and deploy the resulting policy in the real world. This approach is however not always viable if simulated sensor observations differ too much from their real-world counterparts.

In this section, we propose an alternative to this sim2real approach. The presented solution consists of utilizing a digital twin. A digital twin is a digital representation of a real-world concept. In this section, we utilize a digital twin of a tractor that is kept in sync with its real-world counterpart. The main input from the real world was the signal from the Global Navigation Satellite System (GNSS) receiver (Septentrio AsteRx-U) on the tractor, which was then mapped to the digital twin coordinates system. The GNSS had a dual antenna setup which could then provide the heading of the platform as well. Using a cloud-based service, updates were provided in real-time to the digital twin environment to exactly position the simulated tractor as the real-world counterpart. The output from the simulator was the suggested trajectory to the goal pose.

A digital twin of the environment was constructed through constructing a 3D scan of the testing ground. Through utilizing a digital twin the sensors utilized during training, and the actual deployment of the policy are the same, so the sim2real gap is avoided. In order to test this approach a PointGoal navigation agent was trained and deployed (Figure 8.3).



Figuur 8.4: **DD-PPO Architecture**

8.3.1 Architecture and Training

The presented RL approach makes use of the Decentralized Distributed Proximal Policy Optimization (DD-PPO) architecture (Wijmans et al., 2020). The RL approach is able to map high-dimensional inputs to discrete actions. The DD-PPO model consists of a visual pipeline, for which in our case we use a ResNet18 (He et al., 2016).

The resulting learned visual representation is concatenated together with a transformation applied on the output of the GNSS sensor. This output is then passed onto a recurring policy consisting of 2 LSTM (Hochreiter and Schmidhuber, 1997) layers. The final outputs of the model consist of a state value estimation and an action distribution from which actions (move forward, turn left, turn right, and done) can be sampled. The done action should be executed by the agent when positioned less than 2 meters from the goal position. As inputs for the model, we tested a single depth camera, a single RGB camera, or a combination of both RGB and depth. We use these sensors as they are cheap and widely available. The camera is positioned on the front of the AGV.

To train the agent we use the improvement in geodesic distance between the agent and the goal position as a dense reward signal. A slack penalty of -0.01 is subtracted on each step, and a termination bonus of 2.5 is awarded upon successfully utilizing the done action. We train the agent entirely in the Habitat simulator (Savva et al., 2019) where a photorealistic scan of the environment is used. This allows the agent to interact with the terrain safely. While in this case, we trained the agent to specifically work on a single environment, DD-PPO also allows generalization to unseen environments, given enough different training environments and training samples.

Sensors	Success Rate	SPL	Avg. Collisions
Blind	0.9194	0.7294	4.3548
RGB	1.0	0.9454	0.4355
Depth	1.0	0.8882	0.1129
Depth+RGB	1.0	0.9272	0.5161

Tabel 8.1: Evaluation results of the trained policy

8.3.2 Empirical Evaluation in Simulation

Figure 8.5 shows the required number of interactions with the environment. These results indicate that in this setting the agent relies mostly on the GNSS sensor, as the blind agent performs reasonably (60% success rate after 5M training interactions). However, by adding either a depth or RGB sensor the agent achieves near-perfect navigation capabilities on the training set after 5M interactions with the simulated environment.

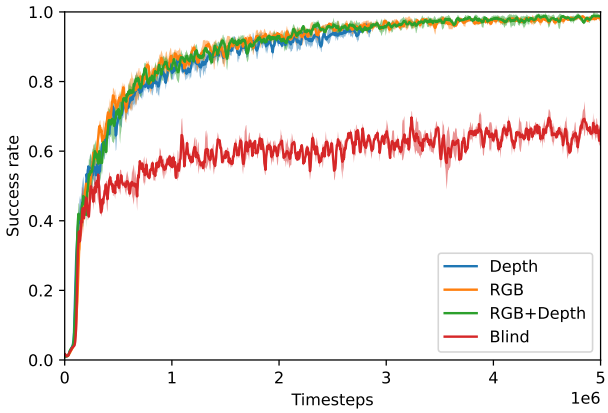
To further evaluate the navigation capabilities of the agent, we created a holdout dataset. This holdout dataset contains goal positions the agent did not see during training. Table 8.1 contains the results of 100 tested episodes. In this table, the success rate indicates the number of episodes the agent could complete successfully. The Shortest-Path Length (SPL) measurement measures binary successes S_i but also considers the length of the path p_i taken, and the shortest path l_i over N episodes.

$$\mathrm{SPL} = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{l_i}{\max\left(p_i, l_i\right)} \tag{8.1}$$

Using no sensors at all (blind), the agent is already able to achieve a strong baseline, however at the cost of often colliding with the environment. Utilizing RGB or a depth camera in this setting has a similar performance. However, using only the depth sensor results in longer paths, but also fewer collisions.

8.3.3 Empirical Evaluation in the Real World

In order to evaluate the applicability of the proposed method we also conducted real-world experiments. In these experiments, a reachable goal position was sampled, and the tractor started from a random position in each episode. To guarantee the safety of involved humans and equipment we opted to utilize a human-in-the-loop setup. Within this setup, the output of the policy (Figure 8.3) was displayed to a human driver, which carried out the actions manually.



Figuur 8.5: **Training performance of the PointGoal agent**
The blind agent can perform basic navigation by relying on the GNSS sensor, however, to further improve to near-perfect results an additional RGB of depth sensor is required to detect and avoid collisions

8.4 Conclusion

In this chapter, two approaches were presented in order to execute PointGoal navigation tasks in real-world environments. In order to attempt a sim2real approach, a simulated version of an office building was constructed. This simulated version can be utilized to test various RL approaches on different navigation tasks. We demonstrated the possibility of training a PointGoal agent within this environment. Ideally, one would be able to directly utilize the in-simulation trained agent in the real world. We, however, noticed that due to small differences, and sources of noise, this is not straightforward.

In order to avoid these issues a second alternative approach was also proposed. In this approach, a digital twin of the agent is utilized which is kept constantly in sync with its real-world counterpart. The state of the digital twin can then be utilized in order to infer actions from the learned policy. This approach was both empirically evaluated in a hold-out region of the simulator, and the real world.

9

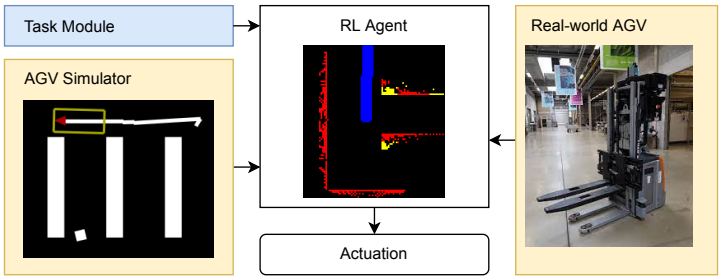
Directed Learned Exploration

The contributions presented in this chapter are based on the publications titled *"A Multi-modal AI Approach For AGVs: A Case Study On Warehouse Automated Inventory"*, and *"Directed Real-World Learned Exploration"*.

9.1 Introduction

AGVs have started to emerge in various industry settings. These vehicles are often utilized to transport various goods from one place to another. In order to perform these tasks, they often rely on navigation systems that are based on markings on the floor or rely on waypoints outlined in static prior maps (Gul et al., 2019; Moshayedi et al., 2021). These systems, however, often rely on highly accurate sensors and dynamics models, require intensive prior configuration, are not able to handle dynamic environments well, and lack robustness (Cadena et al., 2016).

Besides these limitations, it is also often yet unclear how to move beyond pure transportation tasks. For example, if one wants an AGV to find a certain object in the environment (Section 2.6.2.3), heuristic modules which are prone to error propagation, are often required in order to go from object class input to a specific set of navigation world coordinates (Section 2.6.1).



Figuur 9.1: **Directed Exploration approach**
An RL agent is trained in simulation. As the introduced state representation (middle) can be either obtained from the simulator or real-world sensors, the trained policy can be directly used on a real-world AGV platform. In order to direct the exploration, a separate task module is utilized.

In this thesis, we have used RL as an end-to-end learning-based alternative navigation approach in which high-dimensional inputs can be directly utilized in order to output low-level actuation control actions. RL utilizes trial-and-error learning to allow the agent to come up with a policy and task-oriented state representation solely from a reward signal, provided during training. In this manner, RL is capable of implicitly learning the agent affordances and world dynamics, without requiring explicit access to them.

In this chapter, a novel RL approach is presented which goes beyond pure navigation, allowing an AGV to perform various directed exploration (Thrun, 1992) tasks. Such tasks require the agent to actively explore a previously unseen environment in an intelligent (directed) manner. Examples of such tasks might consist of patrolling, searching a specific object, or counting warehouse inventory.

In order to make it feasible to utilize an RL based directed exploration approach in a real-world environment, we propose a novel method for training a directed exploration policy in simulation, which can then be utilized for various downstream real-world exploration tasks, without expensive retraining. Training in a simulated environment allows running multiple environment instances in parallel, allows executing actions safely, and at a much higher frequency than would be possible in the real world.

However, as we demonstrated in Chapter 8, when simulating sensors, inconsistencies between simulated sensors and their real-world counterparts are often unavoidable (sim2real gap). In order to minimize this gap, a LiDAR-based state representation is introduced. We demonstrate that this approach is robust to sensor noise, and thus qualified to bridge the sim2real gap. While LiDAR-based simulations are often very compute intensive (Hanke et al., 2017), the presented approach is less compute-intensive, supporting above real-time simulation.

Similar to the system of instruction enhancements (Chapter 4), the introduced approach of directed exploration offers a loose coupling between the exploration navigation policy and a task module. This loose coupling allows the re-usage of a trained navigation policy in order to perform multiple tasks, without re-training the navigation policy. We demonstrate how a separately supervised trained task module steers the RL policy in order to reduce the uncertainty of its predictions by actively navigating towards better observations.

The presented framework can be applied to a wide range of applications. For the evaluation of the framework, the task of automated warehouse inventory will serve as a use case. In this use case, the uncertainty of an inventory box counter will be utilized in order to direct the exploration.

The contributions of the work outlined in this chapter are the following:

- A novel approach to make learned RL exploration directed through integration with a separately trained task-specific module.
- An embodied training approach capable of (1) learning AGV affordances end-to-end, and (2) balancing directed and more generic exploration.
- A novel representation based on LiDAR point clouds that is able to robustly bridge the sim2real gap between training in simulation and real-world usage.
- A warehouse simulator with procedurally generated warehouse layouts, that can be utilized to further build upon the presented work.

9.2 Directed Exploration Method

9.2.1 Observation Representation

In order to efficiently explore a previously unseen environment, the agent needs a representation of this environment in order to sample an appropriate action. In prior work, depth cameras and RGB cameras have been utilized (Chen et al., 2019; Chaplot et al., 2020b; Burda et al., 2018a; Chaplot et al., 2021) to carry out exploration tasks. However, these sensors are plagued by a large sim2real gap and are often noisy and dependent on environmental conditions (e.g., lighting). A LiDAR sensor is a commonly used environment invariant and less noisy sensor in navigation tasks. The output of a LiDAR is typically represented utilizing a 3D point cloud consisting of coordinates on which the laser encountered an obstacle.

Our method, however, does not require a 3D representation as an egocentric 2D map-like representation is sufficient in order to perform 2D navigation. In order to obtain this 2D representation, the 3D point cloud is projected onto a 2D plane

by flattening the height dimension. Points that are too close, or too far in distance are also ignored in order to mimic the minimum and maximum range of a real-world sensor. While this resembles a laser scan representation, by flattening a 3D representation possible voids in the laser scan due to local holes are avoided.

Additionally, a second input channel containing the local past trajectory of the agent is used in order for the agent to avoid visiting the same place multiple times.

9.2.2 Directed Exploration

In order to direct the exploration behavior of the agent, the proposed architecture allows an additional flattened point cloud map as input. This point cloud should mark regions that have been classified by a separate task module as interesting, and require further exploration. An object detector could for example communicate points of which it has only a low certainty about its classification accuracy. Navigating the AGV close to these points could allow the object detector to improve its classification results by actively obtaining better viewpoints.

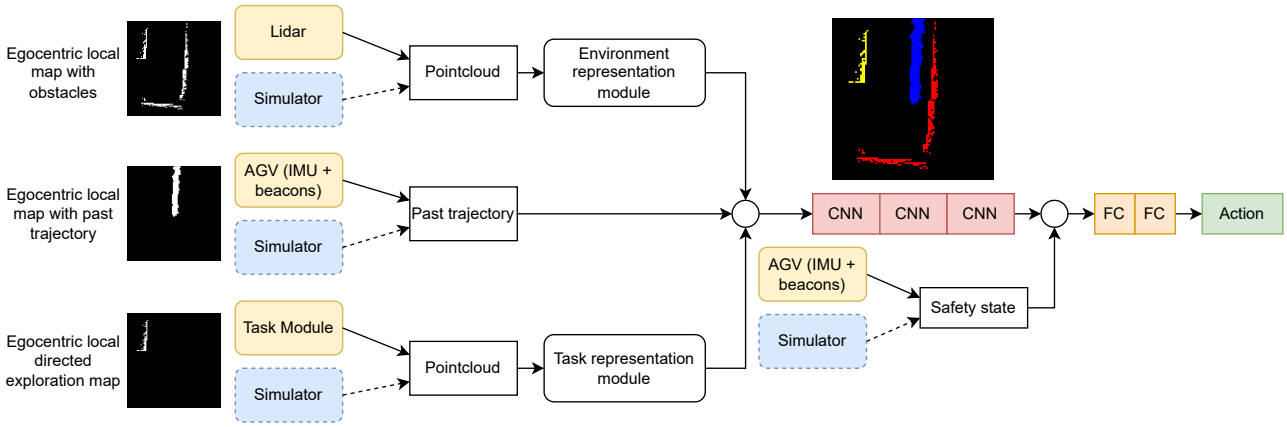
When the task module becomes certain of its prediction (e.g., the classification probability becomes higher than a pre-defined threshold) the points are removed from the directed exploration point cloud.

9.2.3 Policy Architecture

In the presented approach the agent has a visual pipeline that consists of an egocentric local map with obstacles, an egocentric local directed exploration map, and an egocentric local map with the past trajectory of the agent. This visual pipeline is processed by three Convolutional Neural Network (CNN) layers.

In turn, this output is concatenated with a boolean variable which indicates if the safety scanner of the AGV is active in the current state. This safety scanner is an independent system that prevents physical damage due to collisions. This input can be utilized by the policy to maneuver away from obstacles invisible to the visual pipeline. The safety scanner is simulated by checking if the result position of an action would cause a collision before actually moving the simulated AGV.

This concatenated output is then utilized to output a distribution over the discrete set of actions. This distribution of actions can be utilized to stochastically sample actions, leaving room for the exploration of novel strategies. The full architecture is displayed in Figure 9.2.



Figuur 9.2: **Directed exploration agent architecture**
An overview of the architecture of the navigation module. The different inputs are either coming from a real AGV or the simulator. These inputs are processed into an egocentric 3-channel image. Together with the safety state, this can be utilized to directly output actions.

9.2.4 Action Specification

While an RL policy could directly output a continuous value for the steering angle θ and forward velocity V, this is generally regarded as a more demanding setting (Hasselt, 2012). Action discretization has been proposed as a viable less challenging alternative (Marchesini and Farinelli, 2020). We have chosen in our approach to discretize the possible actions into 15 discrete actions:

- Small step forward V=$0.3m/s$
- Reverse straight V=$0.3m/s$
- Large step forward V=$0.5m/s$
- Small δ=$0.17rad/s$, medium δ=$0.4rad/s$, large δ=$0.7rad/s$ turn left/right
- Small δ=$0.17rad/s$, medium δ=$0.4rad/s$, large δ=$0.7rad/s$ reverse turn left/right

These steering angles and velocities have been obtained from human demonstrations collected utilizing a real-world AGV. Actions are executed at 0.5Hz in both the real world and the simulator.

9.2.5 Training

In order to train the agent, the PPO algorithm (Schulman et al., 2017) is utilized. As the reward function, the agent is provided with a small slack penalty of -0.01, which prevents the agent from *slacking off*. In order to study the problem of exploration, three different options for the second term of the reward function are studied:

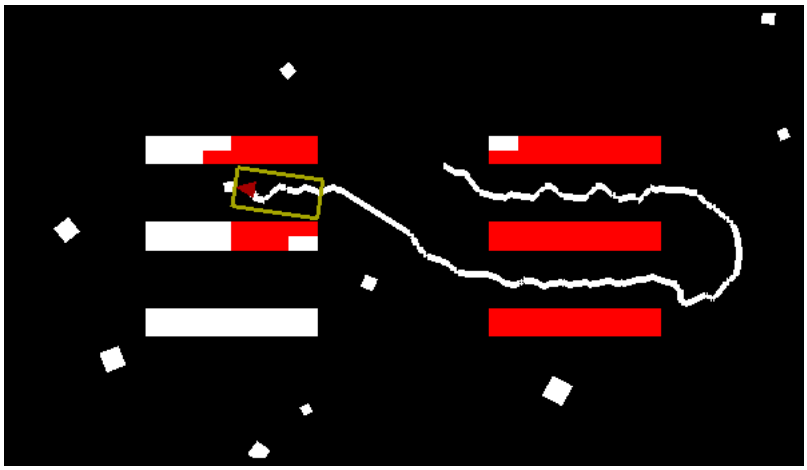
- **Directed**: receive a 0.5 reward when positioning the AGV near an area marked for exploration in the directed exploration point cloud input.
- **Floor coverage**: the environment is divided into a virtual grid with tiles of each $1m^2$. When visiting a new tile the agent receives a positive reward of 0.1.
- **Combined**: in this setting the agent receives both 0.1 for visiting a new tile in the virtual grid, and 0.5 for positioning near areas marked for exploration.

A collision penalty of 0.05 is deducted upon a collision with an obstacle. The different values utilized in the proposed reward functions are obtained through hyperparameter searches.

The agent should be able to function in an environment it did not see before. In order to achieve this behavior the simulated environment is procedurally generated during training. In order to provide the agent with enough instances of seen/unseen trajectories the environment is reset after a fixed amount of 500 steps. In each episode, a variable amount of box obstacles are added in different sizes in order to extend the dynamic navigation capabilities of the agent. Through the introduction of these random obstacles, the agent will be able to handle cluttered real-world environments. The agent is spawned in a random starting position in each episode. A top-down view of an episode can be seen in Figure 9.3. This view is not available to the agent.

9.3 Warehouse Simulator

The simulator was built on top of the *MiniWorld* environment (Chevalier-Boisvert, 2018). It was modified in order to support our customized egocentric point cloud-based representation.



Figuur 9.3: **Top-down view of the simulator**
A top-down view of the simulator is plotted. Racks are plotted in white or red squares depending on whether the agent (yellow rectangle) has explored near them. Through the environment obstacle square boxes of various sizes are added randomly. The agent has no access to this ground truth map.

9.3.1 Layout

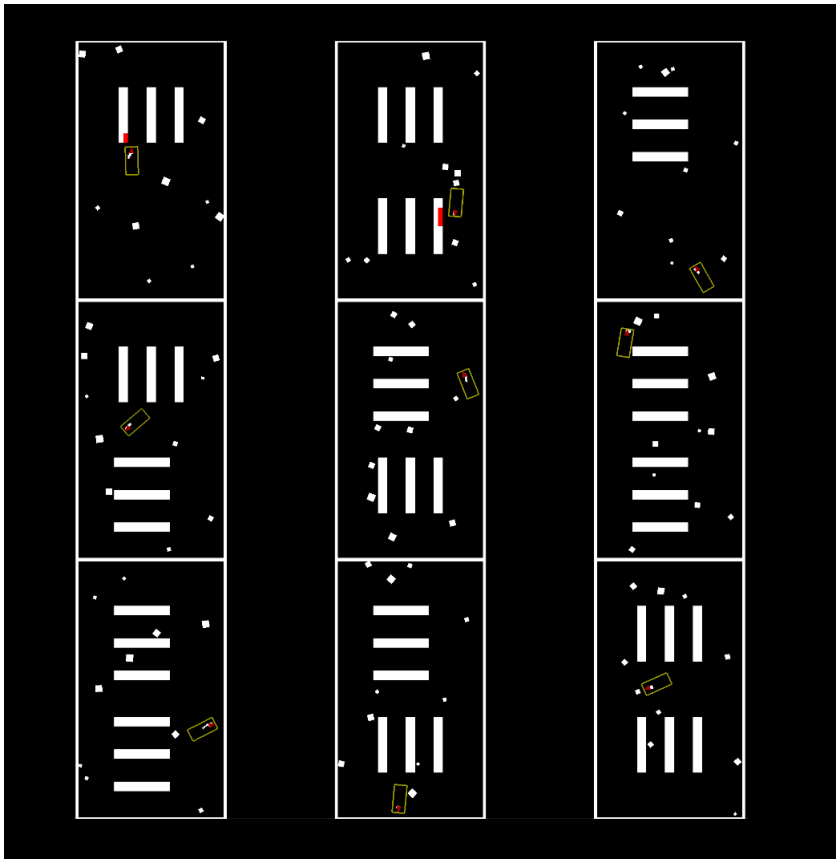
The goal of our simulator is to allow the agent to learn how to navigate in typical warehouse settings. We consider a typical warehouse setting in which multiple rows of racks are placed.

In each episode, the environment is procedurally generated using horizontal racks, vertical racks, or empty spaces. An example containing two sets of horizontal racks is displayed in Figure 9.3. Additional possible layouts are displayed in Figure 9.4.

9.3.2 Lidar Simulation

When using the real LiDAR, the local point cloud can be utilized directly. In the simulator, a ray casting approach is utilized in order to obtain the same representation (as plotted in Figure 9.1).

The proposed observation representation is not only an efficient way of representing the environment in navigation tasks, but it is also a representation that is computationally inexpensive to simulate. This is currently an essential property when choosing an RL-based approach. As RL-based navigation approaches often



Figuur 9.4: **Top-down view of multiple warehouse layouts**

require large amounts of interactions with the environment (Wijmans et al., 2020).

Through utilizing 8 environments in parallel, an average of 250 actions can be executed and evaluated each second on a modest GPU-enabled system (Intel Core i7-9700, Nvidia GTX1060). The real-world AGV in contrast only operates at 0.5 actions per second (on purpose).

9.3.3 Simulated Vehicle Dynamics

While prior work in sim2real and RL has mainly focused on differential drive, the introduced simulator supports Ackerman steering, which is more complex to simulate, but allows the method to be deployed on a wider range of platforms. In order to simulate the movement of the AGV in the simulator, a kinematic bicycle model is utilized, which allows calculating the position of the AGV according to the following set of equations:

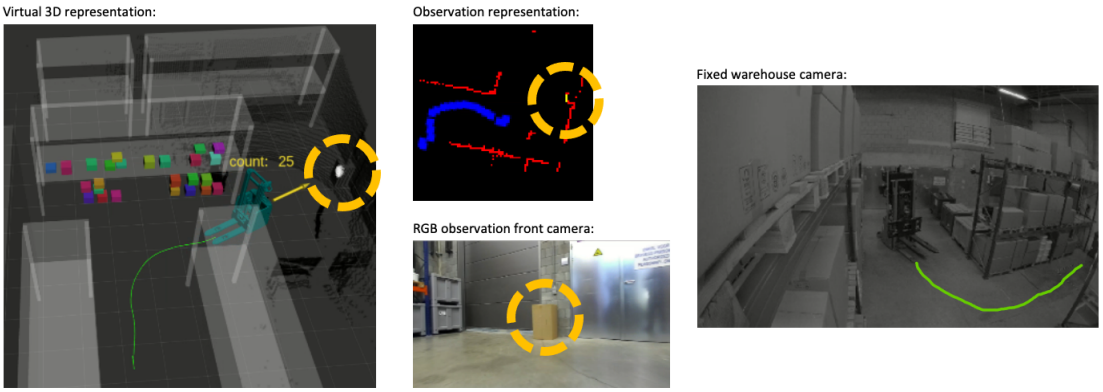
$$\begin{aligned} \dot{x}(t) &= V(t)\cos\theta(t) \\ \dot{y}(t) &= V(t)\sin\theta(t) \\ \dot{\theta}(t) &= \frac{V(t)\tan\delta(t)}{l - a\tan\delta(t)} \end{aligned} \tag{9.1}$$

In this equation, V is the forward velocity, θ represents the yaw angle and δ represents the steering angle. The vehicle specific l is the distance between the front and rear wheels, and a is the lateral distance of the front wheels with respect to the longitudinal center-line. Prior research (Polack et al., 2017) has demonstrated that this type of model can be successfully utilized to produce consistent and feasible trajectories given a limited lateral acceleration. Through embodied trial and error the agent is capable of learning navigational affordances (e.g., can I steer into this narrow corridor from this position).

9.4 Case Study: Warehouse Inventory Task Module

Industrial warehouses are dynamic environments where different assets are changing continuously over time. Automating warehouse management, such as inventory analysis, enables faster operations and fewer errors. There are two main challenges: having a good inventory detector and avoiding counting the same inventory item multiple times.

There are several commercial solutions for automated inventory which are based on having unique identifiers per detected object. In other cases, there is a

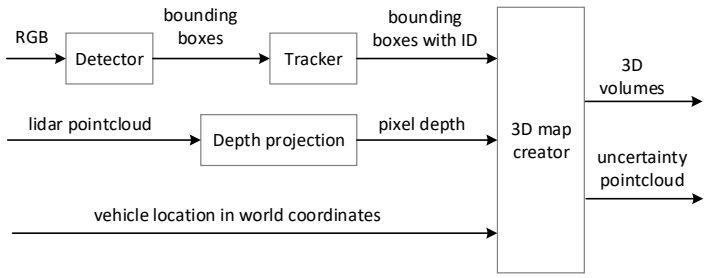


Figuur 9.5: **Visualization of the real-world setup**
In the current state, the task module has counted 25 boxes but is uncertain about the object displayed in front of it (orange dashed circle). The navigation policy is instructed to explore the uncertain object (yellow line in the observation representation) by navigating towards it (yellow arrow). The agent does not have access to the warehouse camera and the 3D environment representation, those are included only for visualization purposes.

predefined route where objects are never re-visited relying on a 2D Multi-Object Tracking (MOT) (Zhang et al., 2021; Du et al., 2022; Cao et al., 2022) that gives unique identifiers between frames. The fusion of LiDAR and camera data for 3D detection (Xu et al., 2018; Mahmoud and Waslander, 2021) and tracking (Zhong et al., 2021) normally relies on annotations for both image and LiDAR data. In contrast, the proposed approach only requires image annotation, which makes it also feasible to incorporate existing datasets.

Directed exploration helps to solve the first challenge, by providing the inventory detector with the capability of actively obtaining better observations through navigation, thus reducing its uncertainty. The second challenge is addressed by constructing a 3D inventory map in world coordinates by combining past detections. The navigation policy will also autonomously steer the AGV to previously unvisited regions in the warehouse in order to allow the 3D inventory map to be fully constructed, and thus allow all boxes to become accounted for.

In Figure 9.5 the entire setup is displayed while executing the inventory task. The principal block is the 3D inventory map creator, which takes the vehicle position, bounding box detections, and per-pixel depth information (aligned with bounding boxes) and provides the detected object's 3D location and size, as well as a point cloud with the uncertain detection areas. The architecture of the perception module is plotted in Figure 9.6.



Figuur 9.6: **Perception module architecture**
Main inputs, outputs, and building blocks of the perception module utilized in the warehouse inventory use case.

9.4.1 Inventory Detection

The detector uses only RGB camera data for classification purposes. LiDAR information is utilized to get and merge the 3D locations of inventory items. A fused approach where LiDAR is also used for detection would require 3D annotations as well. Because these kinds of industrial datasets are not publicly available, and labeling point clouds is an effort that makes the application infeasible in real cases, only RGB annotations were utilized for training the inventory detector. The tracker hands out consistent identifiers to the detections of the same objects in consecutive frames.

In order to run in real-time, the *YoloV7* object detector (Wang et al., 2022) was selected. Starting from a pre-trained version on the COCO dataset (Lin et al., 2014), 4 videos recorded in the testing warehouse have also been annotated in order to fine-tune the object detector. Around 1500 frames were annotated. The detector is trained to learn only the ”box”class. When evaluated upon short recorded trajectories a precision of 89%, and a recall of 85% could be observed. This was however without any directed exploration, which should further improve the observed recall and precision of the model.

9.4.2 Object Tracker

In order to track objects, BYTETrack (Zhang et al., 2021) is utilized, which is a multi-object tracker based on spatial information. It implements a Kalman filter with a constant speed model for the bounding box detection’s position and size and provides two loops where old tracks are matched with new detections, one for high-confidence detections and a second for low-confidence ones. Tracking provides unique identifiers across frames but does not solve the problem of tracking

objects when they re-enter the camera field of view. This is addressed in the 3D map creator.

9.4.3 Object Positions

Although an RGBD camera would already provide per-pixel depth information, after some testing with several depth cameras (Intel Realsense D435, Stereolabs ZED Mini) it was found that the depth accuracy was not high enough, so a 3D LiDAR mounted next to the camera was used instead. The point cloud from the LiDAR is projected on the camera plane, with some radial inflation for closer points to have a richer depth image.

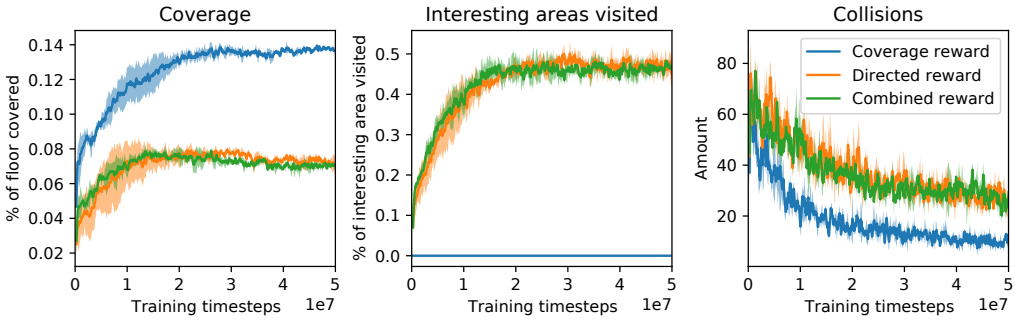
9.4.4 3D Map Creator

This module iterates over the bounding boxes and merges them with the previous map, taking into account the vehicle location. The map contains for each detected box the identifier, confidence, point cloud, centroid, and cuboid size. It also differentiates between certain and uncertain detections. For each detected bounding box, there are two reasons to consider it uncertain:

- Uncertainty in the detector output: If the confidence score provided by the detector is below a certain threshold, then the corresponding object is marked as uncertain.
- Uncertainty in the object location: Using the domain knowledge that boxes have a flat surface, a sample consensus algorithm to fit a plane is computed. In case there are not enough inliers to the plane model, the plane is too far away, or the plane is not seen frontally, then the corresponding object is marked as uncertain.

In the case of certain detections, they need to be merged into the map. There are two possibilities:

- The identifier of the current detection being merged into the map is already the map. In that case, an overlapping comparison is done with all the other detections already on the map, and if there is sufficient overlapping, they are merged. Otherwise, it is merged with the map object with the shared identifier.
- The current detection is not on the map. The same overlapping test is done as in the case above. If there is not enough overlapping it is a new detection



Figuur 9.7: **Training performance in simulation**
Utilizing a coverage-based reward does allow the agent to maximize its coverage, however, it fails to navigate the agent to areas of interest. Utilizing a guiding reward signal allows navigating to the specific regions of interest. A combination of both reward signals has a slightly worse performance for both metrics.

and a new object is initialized in the map. Otherwise, if the overlapping is above the minimum threshold, the new detection is merged into the matched object in the map.

9.5 Empirical Evaluation in Simulation

9.5.1 Training in Simulation

When solely optimizing for coverage, the agent chooses to stay far away from obstacles and avoids going into narrow corridors. Going into narrow corridors has a higher chance of getting stuck, or colliding. The agent has learned to avoid these situations because they lead to a negative reward.

In contrast, when training by specifically utilizing a reward signal encouraging directed exploration, the overall floor coverage of the agent is reduced, but it manages to reach a significant amount of the specified areas of interest within the allocated 500 timesteps. The combination of both reward signals performs best in terms of directional exploration (as measured by the visited marked areas). Figure 9.7 shows the performance during training utilizing the different proposed reward signals.

9.5.2 Evaluation in Simulation

In order to evaluate the performance of the presented approach in simulation, results are collected on a set of 100 fixed warehouse configurations (floor plan, obstacles, starting position) in order to make sure that all runs are equally complex. Each episode is allowed to run for a maximum of 2000 steps, which should be enough for an optimal agent to fully complete the task.

# Obstacles	**Coverage**	**Directed**	**Combined**
0	C=0.46 V=0	C=0.14 V=0.78	C=0.18 V=0.91
5	C=0.34 V=0	C=0.11 V=0.63	C=0.15 V=0.78
10	C=0.27 V=0	C=0.09 V=0.51	C=0.14 V=0.68

Tabel 9.1: Influence of obstacles

When looking at the number of obstacles (Table 9.1) it is clear that the coverage (C) is heavily influenced by the navigational complexity of the environment. The percentage of visited areas (V) marked as interesting does however not remarkably decreases when making the environment more challenging.

In the simulated warehouse, the combined reward signal achieves better results overall in terms of directed exploration results. With no obstacles, the agent is able to visit on average 91% of the areas marked as interesting. While the agent trained using only the directed exploration reward signal is able to visit 78% of the marked areas. The coverage part of the reward signal allows the agent to efficiently navigate in parts of the environment with no areas marked as interesting.

9.5.3 Robustness to Noise

During training, the navigation policies are trained with perfect sensors. As real-world sensors are often far from perfect and are often subject to various kinds of noise, it is important that the navigation policy is able to withstand some amounts of noise added to the sensor observations.

In Table 9.2, the results are plotted when only keeping a certain percentage of the points observed through the LiDAR. This essentially simulates how well the approach can work with a less expensive lower resolution LiDAR. Keeping 0% of the points essentially makes the agent blind.

While this blind agent is not able to cover or explore the environment, the trained

	Coverage	**Directed**	**Combined**
100%	C=0.37 V=0.0	C=0.13 V=0.71	C=0.13 V=0.72
50%	C=0.38 V=0.0	C=0.12 V=0.68	C=0.13 V=0.73
10%	C=0.27 V=0.0	C=0.08 V=0.50	C=0.09 V=0.51
1%	C=0.06 V=0.0	C=0.05 V=0.2	C=0.17 V=0.17
0%	C=0.03 V=0.0	C=0.01 V=0.02	C=0.04 V=0.05

Tabel 9.2: Points sampling (with 3 obstacles)

agents are able to take a significant hit in terms of the number of points coming from the LiDAR input.

Limiting the resolution is however only one source of a potential mismatch between the simulator and the real world. It is also possible that due to a non-perfect accuracy obstacles are detected which in fact are no obstacles, or it could be that obstacles are not detected. In order to simulate these possibilities, and to test the robustness of the trained agents, various amounts of *salt&pepper* noise are added to the input maps.

	Coverage	**Directed**	**Combined**
p=0	C=0.37 V=0.00	C=0.13 V=0.71	C=0.13 V=0.72
p=0.001	C=0.34 V=0.00	C=0.16 V=0.76	C=0.13 V=0.73
p=0.005	C=0.37 vis=0.00	C=0.16 V=0.76	C=0.14 V=0.77
p=0.01	C=0.35 V=0.00	C=0.17 V=0.81	C=0.14 V=0.71
p=0.1	C=0.21 V=0.00	C=0.16 V=0.56	C=0.15 V=0.61
p=0.5	C=0.04 V=0.00	C=0.07 V=0.16	C=0.10 V=0.19

Tabel 9.3: Salt&Pepper noise (with 3 obstacles)

From Table 9.3 it can be concluded that while degrading the accuracy of the observations has a significant impact on the performance, the agent is still able to carry out its task with noisy sensors.

9.5.4 Ablation Study: Trajectory Information

The requirement of adding the past trajectory as input to the agent is an expensive one in the real-world setup, as accurate indoor localization is a challenging problem, often requiring expensive hardware setups.

	Coverage	**Directed**	**Combined**
Enabled	C=0.37 V=0.00	C=0.13 V=0.71	C=0.13 V=0.72
Disabled	C=0.22 V=0.00	C=0.12 V=0.69	C=0.10 V=0.63

Tabel 9.4: Influence of Trajectory Information (with 3 obstacles)

As seen in Table 9.4, removing the past trajectory does have a big impact on the performance of all three studied agents both in terms of coverage and in terms of interesting areas visited.

9.6 Real-world Evaluation

9.6.1 Setup

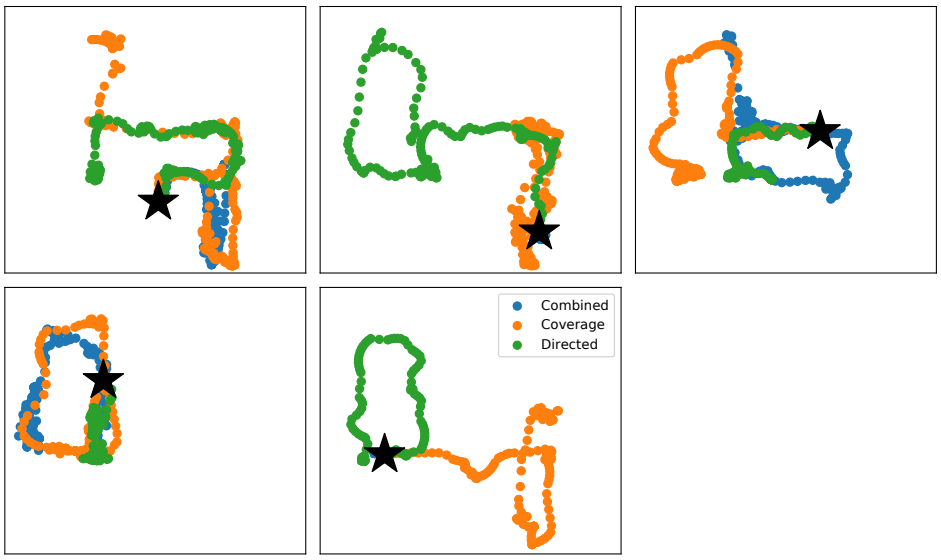
An open experimental platform has been built on top of a standard Still forklift (Figure 9.8). Localization is provided by a commercial system with reflector landmarks with known positions across the warehouse. An Ouster OS1 LiDAR with 64 vertical layers has been used. It has a vertical field of view of 45° and a maximum range of 120m. A Zed mini camera is used for inventory detection. The layout and contents of the warehouse have been kept consistent throughout the evaluation. The AGV was started consistently in different positions throughout the evaluation.

9.6.2 Real-world Performance

In order to evaluate the performance of the approach in a real-world warehouse, 5 positions were selected to run the different trained policies for as long as they made noticeable progress and kept moving. The average results of these 15 runs are presented in Table 9.5. The different trajectories taken by the AGV in each scenario are plotted in Figure 9.9.

In the simulated warehouse, the directed exploration capabilities were evaluated by recording if the AGV would position itself close to the objects marked as interes-

Figuur 9.8: **Real-world AGV setup**
A modified forklift was utilized in order to conduct real-world experiments.



Figuur 9.9: **Trajectories taken by the different policies during real-world evaluation**
The points marked with stars are the fixed starting positions.

	Coverage	**Directed**	**Combined**
Average steps taken	160.4	142.6	106.6
Average coverage	$44m^2$	$33m^2$	$21m^2$
Average reduced uncertainty	26.67	19.67	18.33
Rel. reduced uncertainty	0.61	0.60	0.87

Tabel 9.5: Real world evaluation

ting. In the real-world setting, the directed exploration capabilities are evaluated by recording the reduction in uncertainty of the task module. Concretely, in each run, we track how many objects were marked as certain after they were first marked as uncertain (due to the prediction probability being lower than the specified threshold).

While this is only a small-scale evaluation done in a highly realistic but complex environment filled with obstacles and small passages, some conclusions can be drawn. For example, unsurprisingly, the coverage agent is on average able to cover the largest area of the warehouse. However, by covering a larger area of the warehouse, this agent is also able to achieve the highest absolute uncertainty reduction scores.

When looking at agents which also take the uncertainty of the task module into account (directed, combined) we see that they take radically different trajectories (Figure 9.9). We noticed that these agents often propose more difficult paths, which led to lower floor coverage scores. If we however take efficiency into account the combined agent is able to reduce the most uncertainty per square meter coverage.

9.7 Conclusion

In this chapter, we demonstrated how deep RL can be utilized to perform real-world directed exploration of an unseen environment, relying on egocentric observations coming from a LiDAR sensor, and through training in simulation.

The proposed framework can be utilized in a wide range of different tasks without having to retrain the exploration policy. This was made possible due to the abstract interface introduced between a task-specific module and the navigation policy. As an example, the task of inventory management is studied. Within this study, a supervised trained model was utilized to detect pieces of the inventory. The uncertainty of this model was utilized to steer the exploration behavior of the agent. The novel method is empirically evaluated both in simulation and through utilizing a real-world AGV.

10

Conclusions and Future Perspective

10.1 Review of Problem Statement

Within this thesis, we worked towards extending the capabilities of (real-world) RL approaches by allowing them to make efficient use of abstractions, and to allow existing policies to be adapted. Each of the solutions proposed in this thesis addresses one of the problem statements presented in Section 1.2 as follows:

10.1.1 Towards Human-like Learning

Human intelligence has been a long-lasting source of inspiration for the development of novel AI techniques. The inner workings of human intelligence are however still largely unknown and are still being intensely studied. We do however can clearly identify some common manifesting properties, which might be useful as sources of inspiration when building AI systems.

In the problem statement of this thesis, we proposed to focus on examining how we can harness compositionality and how we can implement learning-to-learn mechanisms. As both topics are still heavily studied in RL, we proposed a few practical solutions within this dissertation:

- **Harness compositionality:** complex problems are often solved by breaking them down into multiple smaller components. For example, when constructing a house, one needs a myriad of different skills and technical plans. An electrician will be in charge of wiring the house, while a bricklayer will raise the walls. In order to make sure that the electrical does not start wiring before there are any walls yet, we often also employ an architect who might not be aware of every brick and wire, but who is managing the big picture.

  This approach is in sharp contrast to how RL systems are currently trained. RL systems typically operate only on one level of abstraction. This often results in enormous search spaces which are difficult to explore. As there was already a vast body of research on the usage of abstractions we started our search on how to harness compositionality by conducting a survey of the literature. The resulting survey was one of the first surveys on HRL which takes both classic methods and novel deep learning-based approaches into account. In addition to a comparative analysis of the major frameworks and approaches we also introduced a set of open challenges.

  Within this thesis, we proposed novel methods which addressed some of these challenges. For example, we addressed subgoal representation through either using language between different layers of a hierarchical approach (instruction enhancement), and we proposed a directional exploration approach in which a separately trained module is able to influence the behavior of a learned exploration policy. We additionally focussed on the formulated challenge of utilizing abstractions for exploration in both the novel SETIE and Disagreement Options framework. Within both these frameworks we either demonstrated progress on tasks on which regular flat approaches were not able to make any progress, or we were able to improve upon the sample efficiency of existing methods. In both cases, compositionality was utilized in order to discover useful modules, and combine these modules in order to develop complex strategies.

- **Learning-to-learn:** one of the challenges which also emerged when conducting our survey on the usage of abstractions within RL was the issue of lifelong learning. While AI systems in general, are often already able to outperform humans in various tasks, they often require comparatively larger amounts of training to do so. This is often caused by the fact that RL systems are typically initialized from scratch, while humans have a vast body of prior knowledge to source from. Within this thesis, we worked on developing various methods which allow an RL policy to adapt prior knowledge, in order to bootstrap novel tasks. In order to facilitate this we demonstrated that the usage of language is an efficient way to assess which prior skills are useful when confronted with a novel task. This was done through either sampling task adaptations, or through the integration with a pre-trained word embedding. Through being able to adapt prior knowledge the proposed methods were able to improve the sample complexity of various navigation tasks.

10.1.2 Efficiently Obtaining the Right Experiences

The second open problem on which this thesis focused, revolved around the question of how a learning system can obtain its own learning experiences efficiently. In order to further advance this longstanding issue, multiple sub-problems were identified and studied:

- **Environment representations:** in navigation problems the usage of maps has been a common approach to representing the environment to the agent. Such map-based representations often depend on human-designed heuristics. While in practice they are often very able to move an agent from point A to point B in a static environment, they also have their limitations. The representation might not give the agent the best view of its actual affordances. And due to its geometric-based approach, the agent is also often not able to utilize semantic clues or perform semantic tasks.

  In this thesis, we fully adopted an end-to-end learned representation approach in simulated environments. Instead of relying on heuristics the proposed methods are able to craft their own task-driven representations of the world, through trial-and-error learning. We were able to demonstrate that in a lot of cases, an agent only equipped with an egocentric RGB camera, and no map or position information is perfectly able to navigate seen and unseen environments by relying on learned state representations.

  However, one major limitation of end-to-end learned representations, is that they often overfit the sensors, and are hard to transfer to different sensors (e.g., a real-world camera). In order to alleviate this issue and work towards real-world RL deployments, we proposed a digital twin-based approach that does not suffer from this problem. Through this digital twin, we were able to, deploy a PointGoal policy trained in simulation, on a real-world tractor. We also introduced a LiDAR-based representation which is both not computationally demanding to simulate and robust to noise. In this setting, we trained an agent in a novel warehouse simulator and were able to deploy the trained model in a real-world warehouse.

- **Exploration scheme:** While it is clear that RL is heavily dependent on an efficient exploration scheme, what such a scheme might actually look like is less clear. In the survey conducted on the usage of abstraction within RL, we concluded that high-level exploration is one of the main benefits of HRL. This was validated in the SETIE approach, in which high-level exploration makes tasks possible, on which low-level only exploration methods could not make any signification progress. We also introduced a second novel approach, which is capable of exploring through the usage of prior policies. The Disagreement Options approach is both able to exploit prior knowledge, use it as temporal abstractions, and explore new solutions.

- **Sample complexity:** humans often only need a single experience in order to learn something (e.g., touching a hot surface). This is again in sharp contrast to how current RL approaches experience the world. Within the presented research we have demonstrated that the sample efficiency of RL approaches can be greatly improved through the introduction of multiple levels of abstraction. Through the introduction of the SETIE approach we were able to demonstrate that an agent is able to learn more efficiently by reasoning both simultaneously on a floor plan level and a low-level actuator level. However, the bigger impact on sample complexity in this thesis can be situated in the work conducted on allowing an agent to adapt its prior knowledge in order to solve novel tasks. We demonstrated that in this context task descriptions formatted in language are an excellent way to predict sample efficient transfer capabilities.

10.2 Review of Hypothesis and Research Questions

In this thesis the following hypothesis was formulated: in order to extend the real-world potential of RL approaches, they need to become more sample efficient. This can be done by introducing compositionality through learning with abstractions, and by allowing the agent to efficiently adapt abstracted prior knowledge obtained in the past.

To validate this hypothesis research questions were formulated which through the previously presented research and experiments we are now able to answer:

1. **What are the various approaches in which abstractions are currently utilized in RL? What problems still need to be solved in order to allow efficient usage of abstractions?** Current approach can be classified into three distinct groups: problem-specific methods, options, and goal-conditional approaches. The current open challenges in this field revolve around top-down hierarchical learning, subgoal representation, lifelong learning, high-level exploration, working on more than two levels of abstraction, making RL interpretable, the development of a clear benchmark, and the development of alternative frameworks.
2. **Can a learned hierarchical approach extend the capabilities of semantic navigation agents?** Through learning both a high-level planner and a low-level controller we were able to demonstrate an agent capable of performing tasks in which prior flat methods did not make any progress in an ObjectGoal setting.

3. **How can a policy trained with a single task in mind select prior knowledge in order to perform another related task efficiently?** We found that there are multiple options available in order to select prior knowledge. The first method we found effective resolves around sampling task adaptations, and uses supervised learning in order to learn a predictive model based on the task descriptions formulated in language. The second effective method we proposed and studied utilized a pre-trained word embedding in order to select relevant prior knowledge.

4. **How can the selected prior knowledge be utilized efficiently when solving novel tasks?** In order to actually use a prior policy we introduced a method that calculates a disagreement score between the action distributions of multiple priors. Through doing this, we managed to use prior policies as temporal abstractions, capable of reducing the sample complexity of novel tasks significantly.

5. **How can an RL approach be utilized in the real world when taking into account current sample complexity limitations?** While typically current RL approaches are trained in simulation, it is not completely clear yet how to utilize them on different real-world platforms. We developed two methods in order to actually perform this task. The first method we found to be capable of operating real-world hardware resolved around the usage of a digital twin. The second method utilized a novel LiDAR-based representation, of which we demonstrated its excellent capabilities of both training a policy in a simulated environment and actually using it in the real world.

10.3 Future perspective

The work presented in this thesis advanced the state of RL by both harnessing compositionality and the introduction of learning-to-learn abilities in RL agents. We often looked at human intelligence as a high-level source of inspiration. There have, however, been identified more threats linked to human intelligence than the ones we have taken as inspiration (Lake et al., 2017) which are currently not yet fully understood, and implemented into AI algorithms. A first line of future work could consist of researching how the introduced algorithms could be extended to assist in building causal models of the world, explicitly focussing on explanation and understanding. HRL might be an excellent *explainable AI* solution in this setting, as a flat RL policy is often opaque, a policy based on high-level abstractions will open up the black box. Especially if the hierarchical system makes use of human-interpretable language as its interface for communicating between layers.

The proposed algorithms in this thesis focus on adapting policies in order to solve novel tasks. This is possible because policies capable of solving related tasks will

require a similar understanding of the environment. A future research direction could be to make these common representations and common theories of world dynamics explicit. Making them explicit could allow novel agents to be trained to utilize this *start-up software*. While it has remained a longstanding open question how such start-up software should be implemented, there recently has been a surge in the development of large pre-trained so-called *foundation models*. These enormous models are trained in a self-supervised fashion and have demonstrated to be capable of capturing large amounts of commonsense knowledge. Both large language models (Devlin et al., 2019; Radford et al., 2019), vision models (Rombach et al., 2022) and even multimodal models (Zeng et al., 2022; Driess et al., 2023) have been introduced. In the presented work we utilized word embeddings as start-up software, which can be seen as the precursor of these large language models. An interesting future perspective might be to study how well these large language models are capable of working with abstractions (e.g., composing commonsense plans), and how they can be utilized to adapt policies to unseen tasks and situations.

The presented task-adaptation approaches started from the hypothesis, that in order to solve an unseen task, existing knowledge is available to be adapted. This might not always be the case, we could envision an additional layer constructed on top of our approaches that handles the questions of whether existing useful priors are available, and whether it would make sense to follow an adaptation strategy or start training from scratch. Additionally, we currently have not yet studied how to efficiently handle the scaling of sets with individual policies, or how to efficiently extend the capabilities of a goal-conditional agent.

We started this thesis with a thorough review of the usage of abstractions within the RL framework. We did this in order to come up with a list of open research challenges within this domain. While we have tackled some of these challenges within this thesis, there still remain the issues of increasing the depth level of hierarchical systems, and the search for learning algorithms developed on top of alternative frameworks.

In the experimental environments utilized in this thesis, the agent was always the sole presence. This is in sharp contrast to how humans experience the world. This observation raises both challenges and opportunities. The main challenge will consist of making the agent able to cooperate socially with other agents both human and non-human. Reasoning on a higher level of abstraction might also help here to intuitively understand what other agents are up to. However, keeping track of what other agents tried through observation or communication could be a valuable source of additional information in order to adapt our behavior. For example, when it is freezing outside seeing someone else fall on the sidewalk might adapt your walking behavior to be more careful. This unfortunate agent who fell could help prevent further accidents by communicating that the sidewalks are in bad shape to other agents.

This thesis proposed some approaches in order to utilize trained RL policies in real-world environments. There are still some issues that need to be ironed out in order to obtain a reliable and widely applicable platform. However, once such a platform becomes available the proposed adaptation methods can potentially also be extended to be capable of operating in a real-world environment. The proposed methods could potentially be utilized to adapt to real sensor observations or to adapt to novel tasks through limited high-level structured exploration. We also envision that the usage of cheaper depth cameras (instead of an expensive LiDAR-based setup) could greatly improve the use cases of RL approaches.

While the methods developed in this thesis relied on online RL, there has recently also been a lot of work done on training embodied systems in an offline dataset-based approach. While we discussed in this thesis that such dataset approaches might not be well suited for embodied navigation settings, the development of various offline RL techniques has been somewhat able to overcome these limitations. Distilling abstractions, and learning how to adapt through the usage of real-world datasets has definitely become an interesting future perspective for learning-based navigation systems.

10.4 Conclusion

This dissertation focussed on navigation problems by utilizing the *embodiment hypothesis*, represented as an RL problem. As RL still requires a lot of interaction with its environment it is still impractical to train an agent directly in real-world settings. It might even be intractable to obtain a satisfying policy using current RL approaches due to the sheer amount of required interactions.

To make RL more sample efficient this thesis proposes to utilize abstractions that can be quickly adapted in order to learn new tasks. In Chapter 3 a survey was conducted on the current approaches of utilizing abstractions in RL settings. This resulted in a taxonomy of methods, a comparative analysis of them, and a list of open challenges.

Inspired by these insights a novel hierarchical approach was presented in Chapter 4. Through structured exploration, an approach was developed capable of localizing objects in procedurally generated environments.

In order to allow the developed abstractions to adapt to novel related tasks Chapter 5 of this thesis introduced novel methods to perform task adaptation based on the description of tasks formatted using language. Chapter 5 introduced a supervised learning method based on sampling task adaptations. Chapter 6 proposes an alternative for this sampling method, by relying on a pre-trained word embedding. As selecting prior knowledge is only part of the adaptation problem, Chapter 7

introduces Disagreement Options, which also proposes a novel method for performing the adaptation itself.

The final part of this thesis revolved around introducing existing RL methods in real-world environments. In order to study the complexities of this problem a novel sim2real environment was introduced. As a first step towards real-world RL, a digital twin-based approach was also proposed in Chapter 8. This approach allowed performing PointGoal navigation utilizing a real tractor. Chapter 9 goes a step further and proposes a method in order to perform directed exploration in previously unseen environments without any priors such as a map of the environment.

Bibliografie

Abel, D., Umbanhowar, N., Khetarpal, K., Arumugam, D., Precup, D., and Littman, M. L. (2020). Value Preserving State-Action Abstractions. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, page 24. 64

Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. (2018). Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*. 90, 99, 100

Ammar, H. B., Eaton, E., Taylor, M. E., Mocanu, D., Driessens, K., Weiss, G., and Tuyls, K. (2014). An automated measure of MDP similarity for transfer in reinforcement learning. In *AAAI14*. 129

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*. 47, 107

Anders, J. and Andrew, G. B. (2000). Automated state abstraction for options using the u-tree algorithm. In *NIPS00*. 60

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., and Zamir, A. R. (2018). On Evaluation of Embodied Navigation Agents. *arXiv preprint arXiv:1807.06757*. 50, 52, 53

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). VQA: Visual Question Answering. In *ICCV15*. 20, 44

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5). 65

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, 34(6). 46

Atrey, A., Clary, K., and Jensen, D. (2020). Exploratory Not Explanatory: Counterfactual Analysis of Saliency Maps for Deep Reinforcement Learning. In *ICLR20*. 107

Bacon, P.-L., Harb, J., and Precup, D. (2017). The option-critic architecture. In *AAAI17*. 58, 63, 64, 72, 81, 98, 101, 104, 105, 106

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. (2020). Agent57: Outperforming the Atari Human Benchmark. *arXiv preprint arXiv:2003.13350*. 47

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR15*. 45

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-End Attention-based Large Vocabulary Speech Recognition. *arXiv preprint arXiv:1508.04395*. 44

Bain, M. and Sommut, C. (1999). A framework for behavioural cloning. *Machine intelligence*, 15(15):103. 65

Baird, L. C. (1993). Advantage updating. 81

Bansal, M., Krizhevsky, A., and Ogale, A. (2018). ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. *arXiv preprint arXiv:1812.03079*. 56

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., hunt, J., Mourad, S., Silver, D., and Precup, D. (2019). The option keyboard: Combining skills in reinforcement learning. In *NeurIPS19*. 58

Barto, A. G. and Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77. 59, 66

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 5:834–846. 93

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint arXiv:1612.03801*. 94

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82. 24, 47

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A Distributional Perspective on Reinforcement Learning. In *ICML17*. 40, 47

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47. 46, 79, 82, 92, 94

Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *NIPS16*. 40, 58

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716. 38, 42

Bengio, E., Thomas, V., Pineau, J., Precup, D., and Bengio, Y. (2017). Independently controllable features. *arXiv preprint arXiv:1703.07718*. 85

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36. 59

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum Learning. In *ICML09*. 63, 80, 90, 99, 108

Beyret, B., Shafti, A., and Faisal, A. A. (2019). Dot-to-Dot: Explainable Hierarchical Reinforcement Learning for Robotic Manipulation. In *IROS19*, pages 5014–5019. 58

Botvinick, M. M., Niv, Y., and Barto, A. G. (2009). Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition*, 113(3):262–280. 58

Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time markov decision problems. In *NIPS95*, pages 393–400. 38

Bresson, G., Alsayed, Z., Yu, L., and Glaser, S. (2017). Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220. 49

Brunskill, E. and Li, L. (2014). Pac-inspired option discovery in lifelong reinforcement learning. In *ICML14*, pages 316–324. 70, 105

Bujanca, M., Shi, X., Spear, M., Zhao, P., Lennox, B., and Luján, M. (2021). Robust SLAM Systems: Are We There Yet? In *IROS21*. 49

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018a). Large-Scale Study of Curiosity-Driven Learning. In *ICLR18*. 40, 58, 106, 173

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018b). Exploration by Random Network Distillation. *arXiv preprint arXiv:1810.12894*. 40, 58

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *IEEE Transactions on Robotics*, 32(6):1309–1332. 171

Cao, J., Weng, X., Khirodkar, R., Pang, J., and Kitani, K. (2022). Observation-Centric SORT: Rethinking SORT for Robust Multi-Object Tracking. *arXiv preprint arXiv:2203.14360*. 180

Castro, P. S. and Precup, D. (2010). Using bisimulation for policy transfer in mdps. In *AAAI10*. 60

Castro, P. S. and Precup, D. (2012). Automatic Construction of Temporally Extended Actions for MDPs Using Bisimulation Metrics. In *Recent Advances in Reinforcement Learning*, volume 7188, pages 140–152, Berlin, Heidelberg. Springer Berlin Heidelberg. Series Title: Lecture Notes in Computer Science. 60

Chaplot, D. S., Dalal, M., Gupta, S., Malik, J., and Salakhutdinov, R. (2021). SEAL: Self-supervised Embodied Active Learning using Exploration and 3D Consistency. In *NeurIPS21*. 52, 173

Chaplot, D. S., Gandhi, D., Gupta, A., and Salakhutdinov, R. (2020a). Object Goal Navigation using Goal-Oriented Semantic Exploration. In *NeurIPS20*. 52, 53, 55

Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R. (2020b). Learning to Explore using Active Neural SLAM. In *ICLR20*. 52, 173

Chaplot, D. S., Sathyendra, K. M., Lample, G., and Salakhutdinov, R. (2016). Transfer Deep Reinforcement Learning in 3D Environments: An Empirical Study. In *NIPS Deep Reinforcemente Leaning Workshop 2016*. 138

Chen, T., Gupta, S., and Gupta, A. (2019). Learning Exploration Policies for Navigation. In *ICLR19*. 173

Chen, V., Gupta, A., and Marino, K. (2021). Ask Your Humans: Using Human Instructions to Improve Generalization in Reinforcement Learning. In *ICLR21*. 115

Chevalier-Boisvert, M. (2018). gym-miniworld environment for openai gym. https://github.com/maximecb/gym-miniworld. 117, 140, 155, 176

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. (2019). BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. In *ICLR19*. 112, 127, 128

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP14*. 45, 113

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2019). Quantifying generalization in reinforcement learning. In *ICML19*. 94

Codevilla, F., Santana, E., Lopez, A., and Gaidon, A. (2019). Exploring the Limitations of Behavior Cloning for Autonomous Driving. In *ICCV19*. 56

Comanici, G. and Precup, D. (2010). Optimal Policy Switching Algorithms for Reinforcement Learning. In *AAMAS10*. 72

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org. 94

Cuayáhuitl, H., Renals, S., Lemon, O., and Shimodaira, H. (2010). Evaluation of a hierarchical reinforcement learning spoken dialogue system. *Computer Speech & Language*, 24(2):395–429. 47

da Silva, B. C., Konidaris, G., and Barto, A. G. (2012). Learning parameterized skills. In *ICML12*. 79

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018a). Implicit Quantile Networks for Distributional Reinforcement Learning. In *ICML18*. 40, 47

Dabney, W., Ostrovski, G., Silver, D., and Munos, R. (2018b). Implicit Quantile Networks for Distributional Reinforcement Learning. In *ICML18*. 47

Damien, E., Pierre, G., and Louis, W. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6. 44

Daniel, C., Neumann, G., and Peters, J. (2012). Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281. 81

Daniel, C., Van Hoof, H., Peters, J., and Neumann, G. (2016). Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357. 81

Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018a). Embodied Question Answering. In *CVPR18*, page 20. 53, 58

Das, A., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018b). Neural Modular Control for Embodied Question Answering. In *Proceedings of The 2nd Conference on Robot Learning*. 54

Dasari, S., Ebert, F., Tian, S., Nair, S., Bucher, B., Schmeckpeper, K., Singh, S., Levine, S., and Finn, C. (2019). RoboNet: Large-Scale Multi-Robot Learning. *arXiv preprint arXiv:1910.11215*. 23

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5:613–624. 76, 77

Dayan, P. and Hinton, G. E. (1993). Feudal reinforcement learning. In *NIPS93*. 66, 67, 87, 91, 98, 108

Dean, T. and Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *IJCAI95*. 57

Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de las Casas, D., Donner, C., Fritz, L., Galperti, C., Huber, A., Keeling, J., Tsimpoukelli, M., Kay, J., Merle, A., Moret, J.-M., Noury, S., Pesamosca, F., Pfau, D., Sauter, O., Sommariva, C., Coda, S., Duval, B., Fasoli, A., Kohli, P., Kavukcuoglu, K., Hassabis, D., and Riedmiller, M. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419. 24, 47

Deitke, M., VanderBilt, E., Herrasti, A., Weihs, L., Salvador, J., Ehsani, K., Han, W., Kolve, E., Farhadi, A., Kembhavi, A., and Mottaghi, R. (2022). ProcTHOR: Large-Scale Embodied AI Using Procedural Generation. In *NeurIPS22*. 52

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR09*. 20

Desai, S. S. and Lee, S. (2021). Auxiliary Tasks for Efficient Learning of Point-Goal Navigation. In *WACV21*. 50

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, page 16. 44, 194

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. https://github.com/openai/baselines. 108

Dibya, G., Abhishek, G., and Sergey, L. (2019). Learning actionable representations with goal-conditioned policies. In *ICLR19*. 92

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13. 68, 74, 85, 93, 101

Dietterich, T. G., Lathrop, R. H., and Lozano-Pérez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2):31–71. 76

Digney, B. L. (1998). Learning hierarchical control structures for multiple tasks and changing environments. In *Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats*, volume 5. 75

Diuk, C., Cohen, A., and Littman, M. L. (2008). An object-oriented representation for efficient reinforcement learning. In *ICML08*. 44, 60, 109

Diuk, C., Cohen, A., and Littman, M. L. (2009). *An Object-Oriented Representation for Efficient Reinforcement Learning*. PhD thesis, ACM Press, Helsinki, Finland. 21

Donahue, J. and Simonyan, K. (2019). Large Scale Adversarial Representation Learning. *arXiv preprint arXiv:1907.02544*. 44, 46

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR21*. 44, 45

Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural computation*, 12(1):219–245. 48

Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., Zeng, A., Mordatch, I., and Florence, P. (2023). PaLM-E: An Embodied Multimodal Language Model. *arXiv preprint arXiv:2303.03378*. 44, 194

Du, Y., Song, Y., Yang, B., and Zhao, Y. (2022). StrongSORT: Make DeepSORT Great Again. 180

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016a). Benchmarking deep reinforcement learning for continuous control. In *ICML16*. 95, 108

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016b). RL$\hat{2}$: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv preprint arXiv:1611.02779*. 82

Dulac-Arnold, G., Mankowitz, D., and Hester, T. (2019). Challenges of Real-World Reinforcement Learning. In *Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36 Th International Conference on Machine Learning*, page 14. 25, 164

Eckstein, M. K. and Collins, A. G. E. (2019). Computational evidence for hierarchically structured reinforcement learning in humans. *Proceedings of the National Academy of Sciences*, 117:29381 – 29389. 58

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep rl: A case study on ppo and trpo. In *ICLR20*. 97

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2019). Diversity is All You Need: Learning Skills without a Reward Function. In *ICLR19*. 64, 89, 99, 106

Ferns, N., Panangaden, P., and Precup, D. (2004). Metrics for finite markov decision processes. In *UAI04*. 60

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML17*. 82

Florensa, C., Duan, Y., and Abbeel, P. (2017). Stochastic neural networks for hierarchical reinforcement learning. In *ICLR17*. 88, 92

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2018). Noisy Networks for Exploration. In *ICLR18*. 40

Fox, R., Krishnan, S., Stoica, I., and Goldberg, K. (2017). Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*. 65, 107

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., Pineau, J., et al. (2018). An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354. 46

François-Lavet, V., Taralla, D., Ernst, D., and Fonteneau, R. (2016). Deep reinforcement learning solutions for energy microgrids management. In *European Workshop on Reinforcement Learning (EWRL 2016)*. 47

Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. (2018). Meta learning shared hierarchies. In *ICLR18*. 82, 101, 105

Fu, J., Kumar, A., Soh, M., and Levine, S. (2019). Diagnosing Bottlenecks in Deep Q-learning Algorithms. In *ICML19*. 23

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-Policy Deep Reinforcement Learning without Exploration. In *ICML19*. 23

Fulda, N., Ricks, D., Murdoch, B., and Wingate, D. (2017). What can you do with a rock? Affordance extraction via word embeddings. In *IJCAI17*. 149

García, J., Fern, and o Fernández (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480. 48

Gatys, L. A., Ecker, A. S., and Bethge, M. (2015). A Neural Algorithm of Artistic Style. *arXiv preprint arXiv:1508.06576*. 44

Georgievski, I. and Aiello, M. (2015). Htn planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156. 57

Ghavamzadeh, M. and Mahadevan, S. (2001). Continuous-time hierarchical reinforcement learning. In *ICML01*. 68

Girshick, R. (2015). Fast R-CNN. In *ICCV15*. 44

Giunchiglia, F. and Walsh, T. (1992). A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389. 26

Goel, S. and Huber, M. (2003). Subgoal discovery for hierarchical reinforcement learning using learned policies. In *FLAIRS Conference*. 77

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In *NIPS14*. 46

Goyal, P., Niekum, S., and Mooney, R. J. (2019). Using Natural Language for Reward Shaping in Reinforcement Learning. In *IJCAI19*. 127

Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. (2017). Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. In *CVPR17*. 21

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. In *ICML15*. 79

Gregor, K., Rezende, D. J., and Wierstra, D. (2016). Variational intrinsic control. *arXiv preprint arXiv:1611.07507*. 88, 154

Greydanus, S., Koul, A., Dodge, J., and Fern, A. (2018). Visualizing and Understanding Atari Agents. In *ICML18*. 107

Gu, J., Chaplot, D. S., Su, H., and Malik, J. (2022). Multi-skill Mobile Manipulation for Object Rearrangement. *arXiv preprint arXiv:2209.02778*. 54

Gu, S., Lillicrap, T., Ghahramani, Z., Turner, R. E., and Levine, S. (2017). Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic. In *ICLR17*. 59

Gul, F., Rahiman, W., and Alhady, S. S. N. (2019). A comprehensive study for robot navigation techniques. *Cogent Engineering*, 6(1):1632046. 171

Guo, Z., Thomas, P. S., and Brunskill, E. (2017). Using options and covariance testing for long horizon off-policy policy evaluation. In *NIPS17*. 70

Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. (2018a). Latent space policies for hierarchical reinforcement learning. In *ICML18*. 64, 84, 90, 96, 99, 104, 107

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement Learning with Deep Energy-Based Policies. In *ICML17*. 88

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018b). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML18*. 59, 90

Hahn, M., Chaplot, D., Tulsiani, S., Mukadam, M., Rehg, J. M., and Gupta, A. (2021). No RL, No Simulation: Learning to Navigate without Navigating. In *NeurIPS21*. 51

Hanke, T., Schaermann, A., Geiger, M., Weiler, K., Hirsenkorn, N., Rauch, A., Schneider, S.-A., and Biebl, E. (2017). Generation and validation of virtual point cloud data for automated driving systems. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. 172

Harb, J., Bacon, P.-L., Klissarov, M., and Precup, D. (2018). When waiting is not an option: Learning options with a deliberation cost. In *AAAI18*. 73, 81, 98

Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., and Precup, D. (2019). The termination critic. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2231–2240. 82, 98

Hasselt, H. V. (2012). Reinforcement learning in continuous state and action spaces. In *Reinforcement learning*, pages 207–251. Springer. 175

Hausknecht, M. and Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable MDPs. In *AAAI Fall Symposia 2015*. 45, 141

Hauskrecht, M. (2000). Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research*, 13:33–94. 37

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an Embedding Space for Transferable Robot Skills. In *ICLR18*. 59

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *CVPR16*. 44, 167

Heess, N., Wayne, G., Tassa, Y., Lillicrap, T., Riedmiller, M., and Silver, D. (2016). Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*. 63, 87

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2017). Deep Reinforcement Learning that Matters. *arXiv preprint arXiv:1709.06560*. 97, 107

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with hexq. In *ICML02*. 68

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., et al. (2017). Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*. 127

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *AAAI18*. 47

Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., and van Hasselt, H. (2019). Multi-task Deep Reinforcement Learning with PopArt. In *AAAI19*. 128

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6). 26, 44

Hinton, G. E. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786). 85

Ho, J. and Ermon, S. (2016). Generative Adversarial Imitation Learning. In *NIPS16*. 65, 154

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. 45, 91, 167

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. 141

Honnibal, M., Montani, I., Van Landeghem, S., and Boyd, A. (2020). spaCy: Industrial-strength Natural Language Processing in Python. 149

Houthooft, R., Chen, X., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. (2016). VIME: Variational Information Maximizing Exploration. In *NIPS16*. 40

Howard, R. A. (1960). *Dynamic programming and markov processes.* John Wiley. 35

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. (2021). How to Train Your Robot with Deep Reinforcement Learning; Lessons We've Learned. *The International Journal of Robotics Research*. 25

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865. 47

Jaderberg, M., Mnih, V., Czarnecki, Wojciech, M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *ICLR17*. 86

James, S., Davison, A. J., and Johns, E. (2017). Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task. In *Conference on Robot Learning*. 56

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019). Sim-To-Real via Sim-To-Sim: Data-Efficient Robotic Grasping via Randomized-To-Canonical Adaptation Networks. In *CVPR19*. 56

Jiang, Y., Gu, S., Murphy, K., and Finn, C. (2019). Language as an Abstraction for Hierarchical Deep Reinforcement Learning. In *NeurIPS19*. 108, 115, 128

Jinnai, Y., Abel, D., Hershkowitz, D. E., Littman, M. L., and Konidaris, G. (2019). Finding Options that Minimize Planning Time. In *ICML19*. 64

Jinnai, Y., Park, J. W., Machado, M. C., and Konidaris, G. (2020). Exploration in Reinforcement Learning with Deep Covering Options. In *ICLR20*. 58, 106

Johnson, M., Hofmann, K., Hutton, T., and Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. In *IJCAI16*. 63, 94

Jong, N. K., Hester, T., and Stone, P. (2008). The utility of temporal abstraction in reinforcement learning. In *AAMAS08*. 62

Jonsson, A. and Barto, A. G. (2006). Causal graph based decomposition of factored mdps. *Journal of Machine Learning Research*, 7. 69

Justesen, N., Bontrager, P., Togelius, J., and Risi, S. (2019). Deep learning for video game playing. *IEEE Transactions on Games*. 46

Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., and Batra, D. (2019). Are We Making Real Progress in Simulated Environments? Measuring the Sim2Real Gap in Embodied Visual Navigation. *arXiv preprint arXiv:1912.06321 [cs]*. 164

Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., and Batra, D. (2020). Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance? *IEEE Robotics and Automation Letters*, 5(4):6670–6677. 50

Kaelbling, L. P. (1993). Hierarchical learning in stochastic domains: Preliminary results. In *ICML93*. 75

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134. 37

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., and Levine, S. (2018). QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *arXiv preprint arXiv:1806.10293*. 47

Kamilaris, A. and Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and electronics in agriculture*, 147:70–90. 44

Karkus, P., Cai, S., and Hsu, D. (2021). Differentiable slam-net: Learning particle slam for visual navigation. In *CVPR21*. 112

Karpathy, A. and Fei-Fei, L. (2015). Deep visual-semantic alignments for generating image descriptions. In *CVPR15*. 44

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE. 94

Khandelwal, A., Weihs, L., Mottaghi, R., and Kembhavi, A. (2022). Simple but Effective: CLIP Embeddings for Embodied AI. In *CVPR22*. 50

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *ICLR14*. 45, 90

Kirkpatrick, J., Pascanu, R., Rabinowitz, N. C., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., and Hadsell, R. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences of the United States of America*, 114. 105

Klissarov, M., Bacon, P.-L., Harb, J., and Precup, D. (2017). Learnings options end-to-end for continuous action tasks. In *Hierarchical Reinforcement Learning Workshop (NIPS17)*. 82

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11). 47

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *NIPS00*. 43

Konidaris, G. and Barto, A. G. (2009a). Efficient skill learning using abstraction selection. In *IJCAI09*. 60

Konidaris, G. and Barto, A. G. (2009b). Skill discovery in continuous reinforcement learning domains using skill chaining. In *NIPS09*. 72, 78, 93

Krishnan, S., Fox, R., Stoica, I., and Goldberg, K. Y. (2017). Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning*. 65

Krishnan, S., Garg, A., Liaw, R., Miller, L., Pokorny, F. T., and Goldberg, K. (2016). Hirl: Hierarchical inverse reinforcement learning for long-horizon tasks with delayed rewards. *arXiv preprint arXiv:1604.06508*. 65

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Communications of the ACM*, volume 60. 26, 44

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. B. (2016a). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *NIPS16*. 58, 64, 80, 94

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016b). Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*. 76, 78

Kumar, A., Swersky, K., and Hinton, G. (2017). Feudal Learning for Large Discrete Action Spaces with Recursive Substructure. In *HRL@NIPS 2017*. 67

Kumar, A., Tucker, G., Fu, J., and Levine, S. (2019). Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *NeurIPS19*, page 11. 23

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40. 22, 58, 63, 137, 193

Lakshminarayanan, A. S., Krishnamurthy, R., Kumar, P., and Ravindran, B. (2016). Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*. 78

Larsen, K. G. and Skou, A. (1991). Bisimulation through probabilistic testing. *Information and computation*, 94(1):1–28. 60

Lazaric, A. and Restelli, M. (2011). Transfer from Multiple MDPs. In *NIPS11*. 29

Le, H. M., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and Daumé, H. (2018). Hierarchical imitation and reinforcement learning. In *ICML18*. 65

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. 20, 26, 44, 98

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 45

Lee, S. Y., Choi, S., and Chung, S.-Y. (2019). Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update. In *NeurIPS19*. 47

Legg, S. and Hutter, M. (2007). A Collection of Definitions of Intelligence. *Frontiers in Artificial Intelligence and applications*, 157. 20

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016a). End-to-End Training of Deep Visuomotor Policies. *The Journal of Machine Learning Research*. 47

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *arXiv preprint arXiv:2005.01643*. 56

Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016b). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *The International Journal of Robotics Research*, 37. 47

Levine, S. and Shah, D. (2023). Learning robotic navigation from experience: Principles, methods and recent results. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 378(1869):20210447. 23, 55

Levy, A., Platt, R., and Saenko, K. (2019). Hierarchical reinforcement learning with hindsight. In *ICLR19*. 85, 104, 107

Li, A. C., Florensa, C., Clavera, I., and Abbeel, P. (2020). Sub-policy Adaptation for Hierarchical Reinforcement Learning. In *ICLR20*. 82

Li, L., Walsh, T. J., and Littman, M. L. (2006). Towards a Unified Theory of State Abstraction for MDPs. In *ISAIM2016*, page 10. 60

Li, Y. (2017). Deep Reinforcement Learning: An Overview. *arXiv preprint arXiv:1701.07274*. 46

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR16*. 47

Lin, K., Zhao, R., Xu, Z., and Zhou, J. (2018). Efficient large-scale fleet management via multi-agent deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1774–1783. 47

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321. 47

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *ECCV14*. 20, 181

Lin, Y., McPhee, J., and Azad, N. L. (2021). Comparison of Deep Reinforcement Learning and Model Predictive Control for Adaptive Cruise Control. *IEEE Transactions on Intelligent Vehicles*, 6(2):221–231. 49

Litjens, G. J. S., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., van der Laak, J., van Ginneken, B., and Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88. 44

Lu, K., Grover, A., Abbeel, P., and Mordatch, I. (2020). Reset-Free Lifelong Learning with Skill-Space Planning. In *ICLR21*. 48

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. (2019). A Survey of Reinforcement Learning Informed by Natural Language. In *IJCAI19*. 46, 127

Machado, M. C., Bellemare, M. G., and Bowling, M. H. (2017). A laplacian framework for option discovery in reinforcement learning. In *ICML17*. 64, 77

Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., and Campbell, M. (2018). Eigenoption discovery through the deep successor representation. In *ICLR18*. 77

Maes, P. and Brooks, R. A. (1990). Learning to coordinate behaviors. In *AAAI90*. 64

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Recent advances in reinforcement Learning*, pages 159–195. 36

Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. In *ICML05*. 77

Mahadevan, S. and Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311–365. 64

Mahadevan, S., Das, T. K., and Gosavi, A. (1997). Self-improving factory simulation using continuous-time average-reward reinforcement learning. In *ICML97*. 38

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking Reinforcement Learning Algorithms on Real-World Robots. In *CoRL18*. 47

Mahmoud, A. and Waslander, S. L. (2021). Sequential fusion via bounding box and motion pointpainting for 3d objection detection. In *2021 18th Conference on Robots and Vision (CRV)*, pages 9–16. 180

Maksymets, O., Cartillier, V., Gokaslan, A., Wijmans, E., Galuba, W., Lee, S., and Batra, D. (2021). THDA: Treasure Hunt Data Augmentation for Semantic Navigation. In *ICCV21*. 52, 53

Mandel, T., Liu, Y.-E., Levine, S., Brunskill, E., and Popovic, Z. (2014). Offline Policy Evaluation Across Representations with Applications to Educational Games. In *AAMAS14*. 47

Mankowitz, D. J., Mann, T. A., and Mannor, S. (2014). Time regularized interrupting options. In *ICML14*. 73

Mann, T. A. and Mannor, S. (2014). Scaling up approximate value iteration with options: Better policies with fewer iterations. In *ICML14*. 70

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56. 47

Marchesini, E. and Farinelli, A. (2020). Discrete deep reinforcement learning for mapless navigation. *ICRA20*. 175

Maron, O. and Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In *NIPS98*. 76

McCarthy, J., Minsky, M., and Rochester, N. (1955). A proposal for the dartmouth summer research project on artificial intelligence. 20, 26, 58

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In *ICML01*. 64, 72, 75, 76, 99, 106

McGovern, A., Sutton, R. S., and Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. In *Grace Hopper celebration of women in computing*, volume 1317. 61

Mehta, N., Ray, S., Tadepalli, P., and Dietterich, T. G. (2008). Automatic discovery and transfer of maxq hierarchies. In *ICML08*. 69

Menache, I., Mannor, S., and Shimkin, N. (2002). Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *ECML02*. 77

Menashe, J. and Stone, P. (2019). Escape room: A configurable testbed for hierarchical reinforcement learning. In *AAMAS19*. 93

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshop*. 141, 149

Mirhoseini, A., Goldie, A., Pham, H., Steiner, B., Le, Q. V., and Dean, J. (2018). A hierarchical model for device placement. In *ICLR18*. 47

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Bae, S., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Babu, A., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J. (2020). Chip Placement with Deep Reinforcement Learning. *arXiv preprint arXiv:2004.10746*. 24

Mishkin, D., Dosovitskiy, A., and Koltun, V. (2019). Benchmarking Classic and Learned Navigation in Complex 3D Environments. *arXiv preprint arXiv:1901.10915*. 21, 49, 112

Misra, D., Langford, J., and Artzi, Y. (2017). Mapping instructions and visual observations to actions with reinforcement learning. *arXiv preprint arXiv:1704.08795*. 127

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *ICML16*. 45, 86, 91

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540). 24, 46, 80, 82, 132, 141

Mo, K., Li, H., Lin, Z., and Lee, J.-Y. (2018). The AdobeIndoorNav Dataset: Towards Deep Reinforcement Learning based Real-world Indoor Robot Visual Navigation. *arXiv preprint arXiv:1802.08824*. 23

Moshayedi, A. J., Xu, G., Liao, L., and Kolahdooz, A. (2021). Gentle survey on mir industrial service robots: Review & design. *J. Mod. Process. Manuf. Prod*, 10(1):31–50. 171

Nachum, O., Gu, S., Lee, H., and Levine, S. (2018a). Near-optimal representation learning for hierarchical reinforcement learning. In *NeurIPS18*. 88, 92, 105

Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018b). Data-efficient hierarchical reinforcement learning. In *NeurIPS18*. 58, 64, 84, 91, 101, 104, 106

Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. (2019). Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning? *arXiv preprint arXiv:1909.10618*. 58, 62, 106

Narasimhan, M., Wijmans, E., Chen, X., Darrell, T., Batra, D., Parikh, D., and Singh, A. (2020). Seeing the Un-Scene: Learning Amodal Semantic Maps for Room Navigation. In *ECCV20*. 53

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. (2020). Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. *arXiv preprint arXiv:2003.04960*. 144

Neumann, G., Maass, W., and Peters, J. (2009). Learning complex motions by sequencing simpler motion templates. In *ICML09*. 72, 79

Ng, A. Y., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML99*. 25, 58, 65, 87, 99, 127

Ng, A. Y., Kim, H. J., Jordan, M. I., and Sastry, S. (2003). Autonomous Helicopter Flight via Reinforcement Learning. In *NIPS03*. 47

Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *ICML00*. 65

Ocana, J. M. C., Capobianco, R., and Nardi, D. (2023). An Overview of Environmental Features that Impact Deep Reinforcement Learning in Sparse-Reward Domains. *Journal of Artificial Intelligence Research*, 76:1181–1218. 58

OpenAI (2018). Openai five. https://blog.openai.com/openai-five/. 47

OpenAI (2019a). Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680*. 24, 47

OpenAI (2019b). Learning Dexterous In-Hand Manipulation. *arXiv preprint arXiv:1808.00177*. 24, 47

Ormrod, J. E. (1999). *Human learning*. Merrill Upper Saddle River, NJ. 24

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepezvari, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., and Van Hasselt, H. (2019). Behaviour Suite for Reinforcement Learning. *arXiv preprint arXiv:1908.03568*. 108

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2022). Training language models to follow instructions with human feedback. In *NeurIPS22*. 44

Pan, X., Ohn-Bar, E., Rhinehart, N., Xu, Y., Shen, Y., and Kitani, K. M. (2018). Human-interactive subgoal supervision for efficient inverse reinforcement learning. In *AAMAS18*. 65

Pan, X., You, Y., Wang, Z., and Lu, C. (2017). Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*. 47

Parr, R. and Russell, S. J. (1998a). Reinforcement learning with hierarchies of machines. In *NIPS98*. 67

Parr, R. E. and Russell, S. (1998b). *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley Berkeley, CA. 38

Partsey, R., Wijmans, E., Yokoyama, N., Dobosevych, O., Batra, D., and Maksymets, O. (2022). Is Mapping Necessary for Realistic PointGoal Navigation? In *CVPR2022*. 23, 50

Pashenkova, E., Rish, I., and Dechter, R. (1996). Value iteration and policy iteration algorithms for markov decision problem. In *AAAI'96: Workshop on Structural Issues in Planning and Temporal Reasoning*. 39

Patel, V. M., Gopalan, R., Li, R., and Chellappa, R. (2015). Visual Domain Adaptation: A survey of recent advances. *IEEE Signal Processing Magazine*, 32(3):53–69. 56

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven Exploration by Self-supervised Prediction. In *ICML17*. 58

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4). 65

Peters, J., Mülling, K., and Altun, Y. (2010). Relative entropy policy search. In *AAAI*. Atlanta. 81

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter Space Noise for Exploration. In *ICLR18*. 40

Polack, P., Altché, F., d'Andréa Novel, B., and de La Fortelle, A. (2017). The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE. 179

Pomerleau, D. A. (1988). ALVINN: An Autonomous Land Vehicle in a Neural Network. In *NIPS88*. 56

Precup, D. and Sutton, R. S. (1997). Multi-time models for temporally abstract planning. In *NIPS97*. 70

Puterman, M. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. New York, NY, USA. 35

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434*. 44

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. Technical report, OpenAI. 44, 194

Radford, N. (1990). Learning stochastic feedforward networks. *Department of Computer Science University of Toronto*, 64(9). 88

Rajendran, J., Srinivas, A., Khapra, M. M., Prasanna, P., and Ravindran, B. (2017). Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain. In *ICLR17*. 29

Ramakrishnan, S. K., Chaplot, D. S., Al-Halah, Z., Malik, J., and Grauman, K. (2022). PONI: Potential Functions for ObjectGoal Navigation with Interaction-free Learning. In *CVPR22*. 53

Ramakrishnan, S. K., Gokaslan, A., Wijmans, E., Maksymets, O., Clegg, A., Turner, J., Undersander, E., Galuba, W., Westbury, A., Chang, A. X., Savva, M., Zhao, Y., and Batra, D. (2021). Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI. *arXiv preprint arXiv:2109.08238*. 51

Ramrakhya, R., Undersander, E., Batra, D., and Das, A. (2022). Habitat-Web: Learning Embodied Object-Search Strategies from Human Demonstrations at Scale. In *CVPR22*. 21, 53, 55

Ramstedt, S. and Pal, C. (2019). Real-time reinforcement learning. In *NeurIPS19*, pages 3073–3082. 48

Rao, K., Harris, C., Irpan, A., Levine, S., Ibarz, J., and Khansari, M. (2020). RL-CycleGAN: Reinforcement Learning Aware Simulation-to-Real. In *CVPR20*. 56

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *CVPR16*. 44

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., and de Freitas, N. (2022). A Generalist Agent. *Transactions on Machine Learning Research*. 21

Riedmiller, M. A., Hafner, R., Lampe, T., Neunert, M., Degrave, J., de Wiele, T. V., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing - solving sparse reward tasks from scratch. In *ICML18*. 86

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. In *CVPR22*. 44, 194

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA. PMLR. 65, 154

Rubio, F., Valero, F., and Llopis-Albert, C. (2019). A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2). 48

Ruiz, D. V. and Todt, E. (2021). BEyond observation: An approach for ObjectNav. In *2th Embodied AI Workshop at CVPR 2021*. 53

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science. 46, 85

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG-TR 166, Cambridge University. 41

Rupprecht, C., Ibrahim, C., and Pal, C. J. (2020). Finding and Visualizing Weaknesses of Deep Reinforcement Learning Agents. In *ICLR20*. 107

Rusu, A. A., Colmenarejo, S. G., Gulcehre, C., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *arXiv preprint arXiv:1511.06295*. 80

Sacerdoti, E. D. (1973). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5. 57

Sainath, T. N., Mohamed, A.-r., Kingsbury, B., and Ramabhadran, B. (2013). Deep convolutional neural networks for LVCSR. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. 44

Salge, C., Glackin, C., and Polani, D. (2014). Empowerment–an introduction. In *Guided Self-Organization: Inception*, pages 67–114. Springer. 88

Santiago, O., Gabriel, S., Alberto, U., Florian, R., David, C., and Mike, P. (2013). A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4). 94

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. In *NIPS17*. 109

Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. (2019). Episodic curiosity through reachability. In *ICLR19*. 106

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. (2019). Habitat: A Platform for Embodied AI Research. In *ICCV19*. 51, 112, 119, 160, 165, 167

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *ICML15*. 63

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized Experience Replay. In *ICLR16*. 47

Schmidhuber, J. (2010). Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3). 58

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61. 20, 44

Schneider, S., Baevski, A., Collobert, R., and Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*. 44

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust region policy optimization. In *ICML15*. 47, 88

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 47, 82, 119, 176

Shang, W., Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Learning world graphs to accelerate hierarchical reinforcement learning. *arXiv preprint arXiv:1907.00664*. 108

Sharma, S., Lakshminarayanan, A. S., and Ravindran, B. (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *ICLR17*. 79

Shu, T., Xiong, C., and Socher, R. (2018). Hierarchical and Interpretable Skill Acquisition in Multi-task Reinforcement Learning. In *ICLR18*. 107

Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT press. 48

Silver, D. and Ciosek, K. (2012). Compositional planning using optimal option models. In *ICML12*. 70

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587). 24, 47

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419). 26

Silver, D. L., Yang, Q., and Li, L. (2013). Lifelong machine learning systems: Beyond learning algorithms. In *AAAI Spring Symposium: Lifelong Machine Learning*, volume 13. 58, 63, 81, 91, 136, 141, 148

Simm, G. N. C., Pinsler, R., and Hernández-Lobato, J. M. (2020). Reinforcement Learning for Molecular Design Guided by Quantum Mechanics. In *ICML20*. 24

Şimşek, Ö. and Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *ICML04*. 72, 76

Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *ICML05*. 77

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8(3-4). 64

Smith, L. and Gasser, M. (2005). The Development of Embodied Cognition: Six Lessons from Babies. *Artificial Life*, 11(1-2):13–29. 24

Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68. 49

Sohn, S., Woo, H., Choi, J., and Lee, H. (2020). Meta Reinforcement Learning with Autonomous Inference of Subtask Dependencies. In *ICLR20*. 82, 108

Stolle, M. and Precup, D. (2002). Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. 76

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., and Littman, M. L. (2006). PAC model-free reinforcement learning. In *ICML06*. 47

Sukhbaatar, S., Denton, E., Szlam, A., and Fergus, R. (2018). Learning goal embeddings via self-play for hierarchical reinforcement learning. *arXiv preprint arXiv:1811.09083*. 88

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *NIPS14*, volume 27. 26, 44

Sutton, R. S. (1984). *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts Amherst. AAI8410337. 81

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44. 47

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, Mass. 24, 40, 46, 58

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NIPS00*. 43

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *AAMAS11*. 84, 85

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211. 61, 65, 69, 72, 92

Sutton, R. S., Precup, D., and Singh, S. P. (1998). Intra-option learning about temporally abstract actions. In *ICLR98*. 73

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv preprint arXiv:1602.07261*. 44

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *CVPR15*. 44

Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J., Maestre, N., Mukadam, M., Chaplot, D., Maksymets, O., Gokaslan, A., Vondrus, V., Dharur, S., Meier, F., Galuba, W., Chang, A., Kira, Z., Koltun, V., Malik, J., Savva, M., and Batra, D. (2021). Habitat 2.0: Training Home Assistants to Rearrange their Habitat. In *NeurIPS21*. 54, 94, 112

Szot, A., Yadav, K., Clegg, A., Berges, V.-P., Gokaslan, A., Chang, A., Savva, M., Kira, Z., and Batra, D. (2022). Habitat rearrangement challenge 2022. https://aihabitat.org/challenge/rearrange_2022. 55

Tang, Y. and Salakhutdinov, R. R. (2013). Learning stochastic feedforward neural networks. In *NIPS13*. 88

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., and Riedmiller, M. (2018). DeepMind control suite. *arXiv preprint arXiv:1504.04804*. 94

Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10. 29, 138

Taylor, M. E. and Stone, P. (2011). An Introduction to Intertask Transfer for Reinforcement Learning. *AI Magazine*, 32(1). 59

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2017). A deep hierarchical approach to lifelong learning in minecraft. In *AAAI17*. 58, 63, 64, 70, 80, 99, 105

Thrun, S. (1992). Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA. 172

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS17*. 56

Todorov, E. (2007). Linearly-solvable markov decision problems. In *NIPS07*. 88

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. *IROS12*. 94

Truong, J., Rudolph, M., Yokoyama, N., Chernova, S., Batra, D., and Rai, A. (2022). Rethinking Sim2Real: Lower Fidelity Simulation Leads to Higher Sim2Real Transfer in Navigation. In *Conference on Robot Learning 2022*. 164

Tsitsiklis, J. N. and Van Roy, B. (1996). Analysis of temporal-diffference learning with function approximation. In *NIPS96*. 46

Turing, A. M. (1950). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460. 20

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. In *SSW16*. 44

van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., and Modayil, J. (2018). Deep Reinforcement Learning and the Deadly Triad. *arXiv preprint arXiv:1812.02648*. 46

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. In *AAAI16*. 47

Van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., and Tsang, J. (2017). Hybrid reward architecture for reinforcement learning. In *NIPS17*. 85

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is All you Need. In *NIPS17*. 45

Vezhnevets, A., Mnih, V., Agapiou, J., Osindero, S., Graves, A., Vinyals, O., and Kavukcuoglu, K. (2016). Strategic attentive writer for learning macro-actions. In *NIPS16*. 79

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). FeUdal Networks for Hierarchical Reinforcement Learning. In *ICML17*. 64, 84, 90, 91, 92, 94, 98, 101, 104, 106, 108

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*. 24, 26, 47

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. (2017). Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*. 94

Visús, A., García, J., and Fernández, F. (2021). A taxonomy of similarity metrics for markov decision processes. *arXiv preprint arXiv:2103.04706*. 129

Waissi, G. R. (1994). *Network Flows: Theory, Algorithms, and Applications*. JSTOR. 77

Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. (2022). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv preprint arXiv:2207.02696*. 181

Wang, H., Dong, S., and Shao, L. (2019a). Measuring structural similarities in finite mdps. In *IJCAI19*. 129

Wang, H., Zariphopoulou, T., and Zhou, X. Y. (2020). Reinforcement learning in continuous time and space: A stochastic control approach. *Journal of Machine Learning Research*, 21(198):1–34. 48

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2017a). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*. 82

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017b). Sample Efficient Actor-Critic with Experience Replay. In *ICLR17*. 47, 59, 148

Wang, Z., Dai, Z., Poczos, B., and Carbonell, J. (2019b). Characterizing and Avoiding Negative Transfer. In *CVPR19*. 29

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *ICML16*. 47

Wang, Z., Yu, J., Yu, A. W., Dai, Z., Tsvetkov, Y., and Cao, Y. (2022). SimVLM: Simple Visual Language Model Pretraining with Weak Supervision. In *ICLR22*. 44

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4). 41, 67

Watkins, C. J. C. H. (1989). Learning from delayed rewards. 142

Weihs, L., Salvador, J., Kotar, K., Jain, U., Zeng, K.-H., Mottaghi, R., and Kembhavi, A. (2020). AllenAct: A Framework for Embodied AI Research. In *CoRR2020*. 112

Weischedel, R., Palmer, M., Marcus, M., Hovy, E., Pradhan, S., Ramshaw, L., Xue, N., Taylor, A., Kaufman, J., Franchini, M., El-Bachouti, M., Belvin, R., and Houston, A. (2013). OntoNotes: A Large Training Corpus for Enhanced Processing. 141, 149

Wijmans, E., Essa, I., and Batra, D. (2022a). How to Train PointGoal Navigation Agents on a (Sample and Compute) Budget. In *AAMAS22*. 50

Wijmans, E., Essa, I., and Batra, D. (2022b). VER: Scaling On-Policy RL Leads to the Emergence of Navigation in Embodied Rearrangement. In *NeurIPS22*. 26, 50, 52, 54

Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., and Batra, D. (2020). DD-PPO: Learning Near-Perfect PointGoal Navigators from 2.5 Billion Frames. In *ICLR20*. 50, 51, 52, 112, 165, 167, 179

Wijmans, E., Savva, M., Essa, I., Lee, S., Morcos, A. S., and Batra, D. (2023). Emergence of Maps in the Memories of Blind Navigation Agents. In *ICLR23*. 23

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4). 43

Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. (2018). Building Generalizable Agents with a Realistic and Rich 3D Environment. *arXiv preprint arXiv:1801.02209*. 53

Xu, D., Anguelov, D., and Jain, A. (2018). Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *CVPR18*. 180

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

Ye, J., Batra, D., Das, A., and Wijmans, E. (2021). Auxiliary Tasks and Exploration Enable ObjectNav. *arXiv preprint arXiv:2104.04112*. 52, 54, 58

Ye, J., Batra, D., Wijmans, E., and Das, A. (2020). Auxiliary Tasks Speed Up Learning PointGoal Navigation. *arXiv preprint arXiv:2007.04561*. 50

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *NIPS14*. 59, 137

Yu, Y. (2018). Towards sample efficient reinforcement learning. In *IJCAI18*. 47

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., Shanahan, M., Langston, V., Pascanu, R., Botvinick, M., Vinyals, O., and Battaglia, P. (2019). Deep reinforcement learning with relational inductive biases. In *ICLR19*. 109

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *ECCV14*. 100

Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., Lee, J., Vanhoucke, V., and Florence, P. (2022). Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language. *arXiv preprint arXiv:2204.00598*. 194

Zhang, J., Hao, B., Chen, B., Li, C., Chen, H., and Sun, J. (2019). Hierarchical reinforcement learning for course recommendation in moocs. In *AAAI19*, volume 33, pages 435–442. 47

Zhang, S. and Sutton, R. S. (2017). A deeper look at experience replay. *arXiv preprint arXib:1712.01275*. 47

Zhang, Y., Sun, P., Jiang, Y., Yu, D., Yuan, Z., Luo, P., Liu, W., and Wang, X. (2021). Bytetrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*. 180, 181

Zhao, X., Agrawal, H., Batra, D., and Schwing, A. (2021). The Surprising Effectiveness of Visual Odometry Techniques for Embodied PointGoal Navigation. In *ICCV2021*. 50, 51

Zhong, H., Wang, H., Wu, Z., Zhang, C., Zheng, Y., and Tang, T. (2021). A survey of lidar and camera fusion enhancement. *Procedia Computer Science*, 183:579–588. Proceedings of the 10th International Conference of Information and Communication Technology. 180

Zhu, H., Yu, J., Gupta, A., Shah, D., Hartikainen, K., Singh, A., Kumar, V., and Levine, S. (2020). The Ingredients of Real-World Robotic Reinforcement Learning. In *ICLR20*. 25, 48, 164

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA17*. 51

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI08*. 88