Exploration and Coverage

A Deep Reinforcement Learning Approach

Nguyen Hai Anh

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Exploration and Coverage
A Deep Reinforcement Learning Approach

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Nguyen Hai Anh

March 19, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Exploration and Coverage

by

Nguyen Hai Anh

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: March 19, 2020

Supervisor(s):

prof.dr. R. Babuska

Reader(s):

dr. M. Kok

ir. R. J. Pérez Dattari

Abstract

This work addresses the problem of exploration and coverage using visual inputs. Exploration and coverage is a fundamental problem in mobile robotics, the goal of which is to explore an unknown environment in order to gain vital information. Some of the diverse scenarios and applications in which exploratory robots can have a significant impact include search and rescue missions, environmental monitoring, and space exploration. Specifically, we focus on the aspect of finding areas of interest, also referred to as targets, in the environment. In this thesis, we propose a deep reinforcement learning based approach relying solely on visual observations. In particular, our method builds upon the off-policy, actor-critic, Importance Weighted Actor-Learner Architectures (IMPALA) framework by including a set of novel auxiliary tasks, i.e. Pose Estimation and Local Map Prediction. These auxiliary tasks are inspired by Simultaneous Localization and Mapping (SLAM) approaches to exploratory robotics problems. The intuition is to assist internal representation learning and build locale specific knowledge by teaching the agent to predict its position and orientation, as well as transfer the visual information to information about the its local proximity. Experiments conducted in the DeepMind Lab simulation environment show improved performance over the base IMPALA agent and demonstrate the effectiveness of these auxiliary tasks. Furthermore, we investigate the performance of the agent, trained through various stages of curriculum, compared to a human controlled agent. The trained agent is shown to outperform the human in the majority of tested scenarios.

Table of Contents

Acknowledgements

This thesis not only is the closing chapter to more than a year full of ups and downs, but it also is the end to a five year long journey here at the TU Delft. Half a year ago, I could not have imagined being in this position of closing my academic journey. This journey is only valuable because of all the people that have been part of the process.

My supervisor, prof.dr. R. Babuska. Thank you for the opportunity to conduct this research project under your guidance. Thank you for your support and patience throughout the process. I certainly have not been the easiest of student to work with. Thank you for your time critiquing my work, even in the weekends or during vacation times. Your inputs always encouraged me to be critical and to take that extra step in the thinking process. I still remember those Systeem en Regeltechniek lectures of yours and prof.dr.ir. B. de Schutter. They have inspired me to opt for a Systems and Control Masters in the first place. I believe it has been a very good choice.

Thank you dr. M. Kok and ir. R. J. Pérez Dattari for spending your time reading my work and being part of the graduation committee.

Many thanks to everyone with whom I have had the privilege to build a relationship with throughout this journey and my dear friends who have been with me before it started.

Lastly, while words cannot do things justice, I would like to express my gratitude towards my parents. Thank you for believing in me and thank you for supporting me in any way possible.

Delft, University of Technology — Nguyen Hai Anh
March 19, 2020

Chapter 1

Introduction

In recent years, mobile robots and autonomous vehicles have gained an increasing popularity due to their reduced cost, higher flexibility and improved reliability. They offer solutions to a wide range of practical problems, including inspection and monitoring of areas of interest, search and rescue missions, planetery expeditions, and many more. One of the major research field regarding this topic is that of exploration and coverage. Exploration and coverage denotes the problem of gaining information in an unknown environment through the deployment of search strategies.

Exploration and coverage involves a wide spectrum of different aspects such as data processing and storage, localization, and control. Formally, the exploration aspect refers to the task of searching within an *a priori unknown* environment, whereas the coverage aspect is more of a planning problem and refers to physically visiting an *a priori known* environment such that subtasks can be performed. There is a distinctive overlap between the two aspects as an effective exploration strategy goes hand in hand with a high coverage performance. This work specifically focuses on autonomously searching for areas of interest, also referred to as targets, in unknown environments. The higher the number of targets in the environment, the larger the space the agent needs to cover in order to find all these targets. Time is often a major factor in these applications, e.g. in search and rescue missions it could be the difference between life and death. Hence, an effective exploration strategy is of crucial importance to the success of the approach. Like many real world robotics problems, exploration and coverage comes with numerous challenges. The first major challenge is the uncertainty aspect, native to unknown environments. It restricts the utilization of planning solutions and often results in conservative assumptions, limiting the performance of these methods. Other challenges include sensing capabilities, onboard hardware complexity, computational power, communication and maintainance of communication.

Many works in the literature on this problem proposed a multi-step approach involving some form of frontier based exploration. The idea of frontier based exploration was introduced on Nomad mobile robots equipped with various laser rangefinders, sonars, and infrared sensors [1]. Many other works have since followed in its footsteps [2], [3], [4]. The referred multi-step process particularly involves detecting the boundary between the covered space and the

unexplored space, selecting the subsequent destination for the agent from the list of frontiers, and executing a behavior/path planning algorithm to move towards the chosen destination. A terminal condition is adopted dependent on measures regarding the remaining frontiers. This often involves storing an occupancy grid or graph to keep track of the covered discretized space. Frontier based exploration can be integrated with other aspects of mobile robotics such as topological map building [4], or in a multi robot setting with wireless networking [5]. Despite their successes, the vast majority of these approaches focused on the setting in which robots rely on some form of active range sensors. This can be limiting as we strive towards smaller, lighter, and more energy and cost effective robots.

In this regard, vision based approaches, which rely solely on cameras, have attracted an increased amount of popularity due to their low footprint, in terms of costs, power consumption, and onboard hardware complexity. Many vision based approaches for exploration are related to the Simultaneous Localization and Mapping (SLAM) problem, such as ORB-SLAM [6] and LSD-SLAM [7]. SLAM is the problem of constructing and updating a map of an unknown environment and simultaneously localize the agent within it. SLAM approaches involve many different submodules and can become very complex as one rely on explicit map reconstruction and accurate state estimation through filtering techniques, which is susceptible to drift and uncertainty.

The last decade has witnessed many successes in the field of artificial intelligence, and in particular in reinforcement learning (RL). Superhuman performance has been achieved in arcade computer games [8], [9], and state-of-the-art algorithms have been successfully applied to relatively simple, real world robotics tasks [10]. In this work, we seek to explore the application of RL to the exploration and coverage problem, in the hope that it carries the positive aspects of single camera systems, while offering a simpler approach compared to SLAM by learning end-to-end. Reinforcement learning is a form of machine learning that lies between supervised and unsupervised learning. Unlike supervised learning, where the agent learns the relation between input-output pairs, RL involves an agent interacting with an environment and receiving rewards according to its actions. Over time, by trying to maximize the cumulative reward, the agent adapts and learns to complete a complex task. This process resembles the natural way humans and animals learn through life. With the rapidly evolving computer hardware architectures, combining deep learning and RL allows one to face tougher and more extensive problems than ever before. However, while numerous forms of success have been achieved, many required in the order of hundreds of millions of samples to reach a desired performance. This is one of the major limiting factor of RL as it delays the evaluation process and restricts real world integrations. In this work, we seek to counter the *sample efficiency* problem by incorporating SLAM inspired auxiliary tasks as a regularization tool to improve robustness and training performance. To further enhance the performance of the agent, we incorporate the idea of curriculum learning. Through a gradual increase in the complexity of the problem, we hope to teach the agent more complex skills involved in the exploration and coverage task that the agent may not have learned otherwise.

In the following chapter, we provide the background information to reinforcement learning and deep reinforcement learning, and cover several works in the literature to build the foundation for our approach. Chapter 3 provides the formal setting to the exploration and coverage problem and Chapter 4 provides a detailed description of our proposed approach following the presented formal setting. The details to the evaluation process and the accompanying results are presented and discussed in Chapter 5. Finally, Chapter 6 closes with the key

takeaways from this project and provides the recommendations for future works.

Chapter 2

Preliminaries

In this chapter, we cover the preliminaries of the proposed approach to the exploration and coverage problem. We start by introducing the principles of reinforcement learning as a form of machine learning involving sequential decision making in Section 2-1. Section 2-2 focuses on the the utilization of (deep) neural networks as function approximators and the basics of deep reinforcement learning (DRL). Section 2-3 covers the challenges involved in solving (deep) reinforcement learning problems. The related literature is then highlighted as a foundation for our approach in Section 2-4, 2-5, and 2-6, including resolving partially observable environments, the utilization of auxiliary losses to aid the learning process, and the IMPALA actor-critic architecture for scalable learning.

2-1 Reinforcement Learning

Reinforcement learning is a mechanism in which an *agent* autonomously interacts with an *environment* and adapts itself in order to optimize its behavior for a specific task, based on feedback from the environment. This process is very similar to the adaptation process encountered in humans and animals. It allows organinisms to solve complex problems through exploratory bahavior and through learning from failure and success. One can think of examples such as the process of learning to take the first steps as a child or trying to solve a new puzzle. Due to its trial and error nature, once trained to perform a specific task, an RL agent has the potential to adapt its behavior based on changes in the environment. This distinguishes RL from traditional planning solutions where the behavior of agents are hand engineered by the designer. Furthermore, an RL agent possess the intrinsic ability to deal with uncertainty, which is often not considered in planning algorithms.

The objective of an RL agent is to learn a certain mapping from states to actions, known as the *policy* π, that would allow the agent to optimally interact with the environment. The performance measure is the cumulative reward the agent receives from the environment. It is typically defined as the the discounted sum of immediate rewards, also referred to as *return*, $R_t^\pi = \sum_{t=0}^{N} \gamma^t r_{t+1}$, where $\gamma \in [0, 1]$ is the discount rate regulating the importance of future

rewards, and N is the time horizon of the problem. The immediate rewards r_t, received in each time step t, are scalar values generated by the *reward function*. It is the instantaneous feedback evaluating the transition from state x to state x' through action a. Through the definition of the reward function, the designer hopes to direct the learning process such that the agent learns to accomplish the given task. Figure 2-1 shows a schematic representation of the basic RL framework.

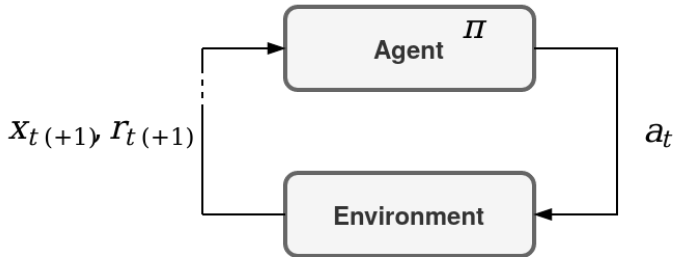


Figure 2-1: A schematic representation of the reinforcement learning framework. At every time step t, the agent decides to take an action based on its policy, resulting in a reward and the transition to a new state.

As the future rewards are (possibly) uncertain, the RL problem relies on the expectation of the cumulative reward to evaluate the behavior of the agent. The *state-value function* V^π is the expected return starting from state x under the policy π. It represents the *policy-based* long-term desirability of a state. Another notion to infer the expected return in RL is the *action-value function* Q^π, which denotes the expected return of following action a in state x, and successively following policy π. The corresponding state-value function V^π and action-value function Q^π are formally defined as:

$$V^\pi(x) = \mathbb{E}^\pi\{R_t|x_t = x\}$$
$$Q^\pi(x,a) = \mathbb{E}^\pi\{R_t|x_t = x, a_t = a\}$$

where $\mathbb{E}^\pi$ denotes the expectation given the policy.

A fundamental aspect of RL is the Markov property. In general, the state of the environment can be represented as a probability distribution based on all past states and actions. The Markov assumption denotes the state being information dense, such that it entails all relevant data of the past. Consequently, one-step dynamics are sufficient to predict the next state and reward. An RL task satisfying the Markov property is called a *Markov Decision Process (MDP)* [11]. An MDP is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ represents the set of states, $\mathcal{A}$ represents the set of actions available to the agent, $\mathcal{P}$ represents the probability of transitioning from state x to a new state x' through action a, $\mathcal{R}$ represents the reward function, and γ represents the discount factor. The solution to an MDP is the policy, defining the behavior in each state. Two of the best known algorithms for solving MDPs are policy iteration [12] and value iteration [13].

An MDP is based on the assumption that the complete state of the environment is observable, which in reality is rarely the case due to factors such as noise and disturbances, or mechanical degradation of components. For example, Pacman can be viewed as an MDP. A poker game on the other hand cannot, since the opponent's cards are unknown. Likewise, the problem of

exploration and coverage is so-called *partially observable* as one aims to manoeuvre within an *a priori unknown* environment by solely relying on visual observations. The RL problem can then be formulated as a *Partially Observable Markov Decision Process (POMDP)* [14], which in addition to the above-mentioned tuple, maintains a probability distribution over the set of possible states, based on a set of observations and observation probabilities.

Hence, an RL problem either seeks to optimize for the policy directly and/or for the value function, based on which the optimal policy is determined. In general, value based methods tend to suffer from poor convergence, as slight changes in the value space may result in drastic deviations in the policy space. As policy based methods work directly in the policy space, smoother learning curves can be obtained. However, they tend to suffer from high variance and poor sample efficiency [15]. An RL algorithm in which both the policy and the value function are optimized is called an *actor-critic* method [16]. Here, the *actor* refers to the learned policy, and the *critic* refers to the corresponding value function, which evaluates the policy adopted by the agent. Actor-critic methods seek to bring the benefits of both value and policy based RL. Typically, the updates of the actor and the critic involve the *temporal difference error*. The temporal difference error refers to the difference between the successive value estimates, formally defined as $\delta_t := r_{t+1} + \gamma V(x_{t+1}) - V(x_t)$. This work mainly focuses on actor-critic reinforcement learning. Figure 2-2 represents the general architecture of an actor-critic method.

For a more elaborate overview of RL, we refer to [17]. For the interested reader in the topic of RL in robotics, we recommend works such as [18] for a comprehensive survey.

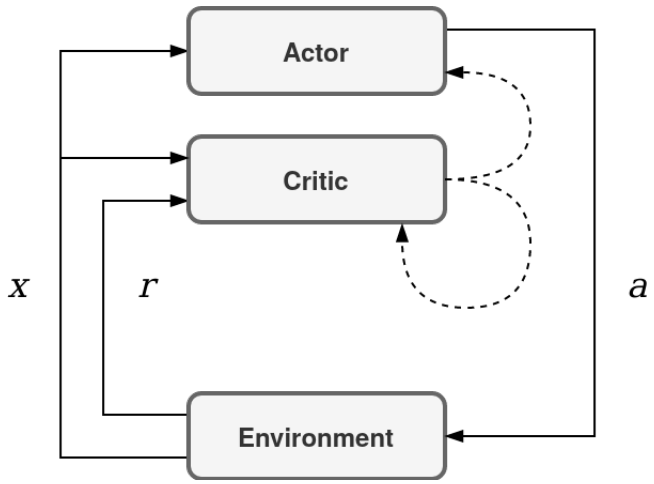


Figure 2-2: A schematic representation of an actor-critic agent. The agent takes an action based on its current policy (actor), resulting in a new state and a reward. This feedback is incorporated in the update rule for both the actor and the critic.

2-2 Function Approximation with Neural Networks

Traditionally, RL methods store and update values corresponding to the states and actions in tabular form. The amount of memory required and the computational expense of sweeping through the table to look up specific state-action combinations grow with the size of the state

and action spaces. Consequently, this becomes problematic for problems with large or continuous state and action spaces. Function approximators can be utilized to address this problem. A *parameterization vector* $\theta = (\theta_1, \theta_2, ..., \theta_n)^T$ is used to parametrize the value function or policy. The function approximator is viewed as a mapping from the parameterization vector to the space of the value function or policy. The complexity decreases as the number of parameters is significantly less than the number of state-action combinations. Transitions from different regions of the state and action spaces activate certain parameters and contribute towards learning these parameters. Consequently, generalization and sample efficiency can be improved.

With the recent advances in deep learning, the use of neural networks as function approximators has become increasingly popular. Neural networks, or *artificial neural networks* (Figure 2-3(a)), are inspired by biological networks that constitute our sensory system and brain. The complex interconnections of neurons allow us to process large amounts of data and extract vital information. Similarly, an artificial neural network consists of *layers* of neurons. Such a network is considered *deep* when it contains two or more *hidden layers* between the input and output layer. This is irrespective of the number of neurons in each layer, known as the *layer size*. Each neuron (Figure 2-3(b)) converts a vector of inputs $\{x_1, x_2, ..., x_n\}$ to an output by computing the weighted sum over the inputs, and passing its biased version through an activation function σ:

$$y_i = \sigma(\sum_{i}^{n} x_i w_{ij} + b_i) \tag{2-1}$$

The activation function introduces nonlinearities, allowing approximations of arbitrary complex functions. More complexity can be introduced by increasing the number of hidden layers or the layer sizes. While this allows the network to approximate more complex functions, it comes at the cost of higher computational loads and the potential of *overfitting*, the modelling error of fitting a particular set of data too closely.

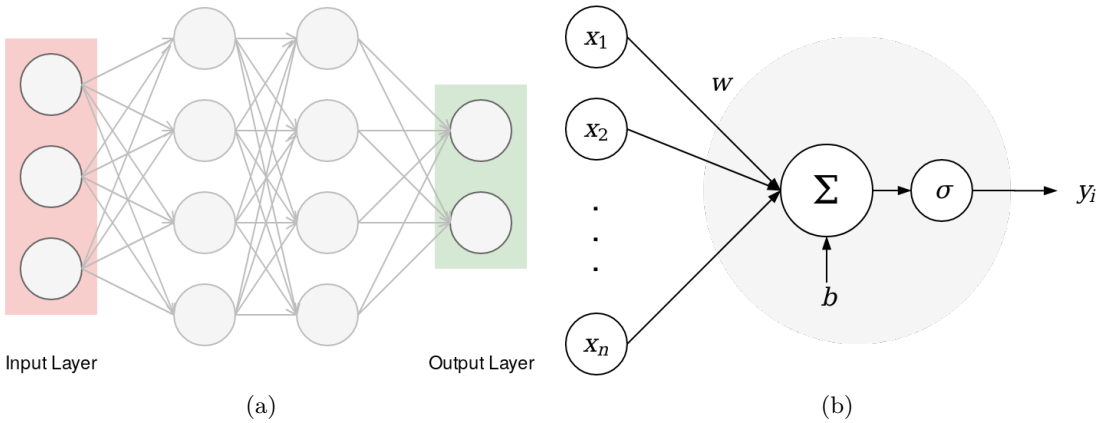


Figure 2-3: (a) A multilayer feedforward neural network with three inputs, two hidden layers, and two outputs, and (b) a representation an artificial neuron, which enforces a nonlinear transformation over the biased weighted sum of the input vector.

The nonlinear transformation is performed for each node from the input layer to the output layer. The objective is to iteratively update the weights of the network such that the residual

between the predicted output y and the desired output can be minimized, based on a certain cost function $J(\theta)$. The parameters θ represent the weights of the network. This iterative process is referred to as *training*. In general, the update is performed through *backpropagation* [19]. The backpropagation technique computes the gradient with respect to the network parameters, and takes a step in the direction of the derivative which decreases the cost: $\theta_{n+1} = \theta_n - \alpha \frac{\partial J}{\partial \theta}$, where α is the learning rate.

Since computing the gradient over the whole training set is computationally expensive, methods such as *stochastic gradient descent* are often applied. The overall gradient is estimated by computing the gradient over randomly sampled batches rather than over the whole set. In addition to the computational benefits, the use of *mini-batches* in training also allows smoother convergence and larger learning rates. The tuning of the *learning rate* α is of high importance in this aspect. Small learning rates result in slow convergence and can cause the problem to get stuck in local minima, while large learning rates result in overshoot and can have an unstable and diverging effect on the learning process. To address this problem, the idea of *adaptively* changing the learning rate rather than utilizing a fixed parameter arose. Common optimizers include `AdaGrad` [20], `Adadelta` [21], `RMSProp` [22], and `ADAM` [23].

2-3 Challenges in Reinforcement Learning

With the rise of DRL, state-of-the-art performances have been achieved in an increasingly large range of problems with various degrees of complexities. Prominant examples include the model-based *AlphaGo* [24] and its improved version *AlphaGo Zero* [25], which achieved super human performance in the game of Go. This, at the time, was viewed as one of the biggest breakthroughs in the field of artificial intelligence (AI) since *Deep Blue* [26] beat grandmaster Garry Kasparov in the game of chess. Other examples include *DQN* (2015) [8], *A3C* (2016) [27] and *SAC* (2018) [10]. Professional player performance was achieved in the classic Atari arcade game selection as well as state-of-the-art performance in navigation tasks in 3D environments and numerous continuous control tasks from the OpenAI Gym environment.

However, many aspects of RL remain particularly challenging such as credit assignment, learning stability, sample efficiency, and partial observability to name a few. For real world problems, reward shaping proofs to be very difficult. Intuitive reward choices likely result in sparse, noisy and delayed signals. Consequently, the learning process becomes inefficient. Works such as [28] and [29] have shown that improper formulations of reward signals can result in totally undesired (and unexpected) behaviors. Delayed reward signals are amongst the causes of the presence of noise in gradient estimators. As a result, the learning curves of DRL methods are often non-monotonic and, in particular, the agent often seems to forget highly rewarded policies. The sparsity of rewards also has great effects on how fast the agent can learn the representations required in order to successfully complete the task. While some environments are relatively simple and can be solved in a small number of samples, many problems reflecting real world scenarios may take in the order of hundreds or thousands of millions of steps to learn an appropriate behavior. Exploration and coverage can be considered a *hard exploration* robotics task and faces these challenges. This work focuses on integrating and extending the remedies proposed in the literature for our specific problem.

2-4 Partial Observability

While MDPs provide a formalism for sequential decision making problems that conveys perfect information for an optimal behavior, many real world problems do not adhere to its assumptions. The partially observable formulation do not adopt the perfect state assumption, but relies on observations that are dependent on the state. The observations are both spatially limited as well as temporally limited. An agent acting in a partially observable world relies on the observations to form a belief about the actual state of the environment. In order to function in such environments, the agent needs to build a capacity for temporal integration of observations. In other words, while a single observation may not convey sufficient information about the state, a sequence of observation over time may.

The literature proposes a number of ways develop the referred temporal integration. One solution is to provide the agent with a set of successive observations rather than a single observation [30]. While this approach is shown to perform well in relatively simple settings, it has several drawbacks. First and foremost, the number of successive observations needed is highly dependent on the problem. Hundreds of past observations may be required to form a desired behavior. Secondly, an experience buffer is required to store successive observations. As a result, the necessary memory grows with the size of the observation and with the number of observations, causing the problem to become intractable.

Another solution is to incorporate the temporal integration within the agent itself. This can be achieved by utilizing recurrent neural network modules [31], [32]. In particular, a *long short-term memory (LSTM)* [33] can be used to form historical context. An LSTM unit has feedback connections, unlike feedforward neural networks. This allows information to be processed sequentially and hidden states to be retained. An LSTM unit typically consists of a cell state, an input gate, an output gate, and a forget gate. The cell state is responsible for retaining information throughout the sequence, while the three gates are regulators controlling which part of the data to retain and which to forget. Recurrency allows the agent to memorize information about past observations and learn the temporal patterns of observations.

2-5 Auxiliary Learning

In many real world problems, extrinsic rewards can often be sparse. Imagine a standing-up problem in which a large variety of ways exists to accomplish the task and rewarding intermediate steps becomes very challenging. Or animals having to cover a vast distance before being able to find food. Traditional reinforcement learning approaches have been designed with the objective of directly maximizing the cumulative reward. Due to sparse reward signals, one often faces challenges in terms of poor sample efficiency and convergence to local minima. The concept of utilizing auxiliary tasks arose to enhance training in these cases. Rather than just directly maximizing the cumulative reward, the agent is augmented with subtasks which are designed to focus on the important aspects of the problem and the related visual features. The auxiliary tasks contribute to the learning process in the form of additional loss terms in the optimization problem.

The idea of auxiliary learning for neural networks was first proposed to avoid local minima [34]. Additional cost terms was later introduced to RL in the form of general value functions

[35]. Rather than regarding the environmental rewards as in the case of the standard value function in RL, these cost terms consider other signals. To address the sample efficiency aspects of DRL, the concept of auxiliary learning was integrated to popular algorithms such as *DRQN* [36] and *A3C* [32]. It is important to note that the objective of these additional tasks is not the actual prediction performance, but rather to be utilized as an enhancement tool for training and learning representations that are crucial to successfully achieve the main objective. In essence, the auxiliary losses serve as a regularization measure to improve robustness, performance and training speed.

In general, the implementation of auxiliary learning is performed by splitting the agent network into multiple heads (Figure 2-4). Besides the main head responsible for the approximation of the policy and/or value function, additional heads each serve their own purpose. The corresponding errors propagate to the shared layers of the network and all contribute to the learning process.

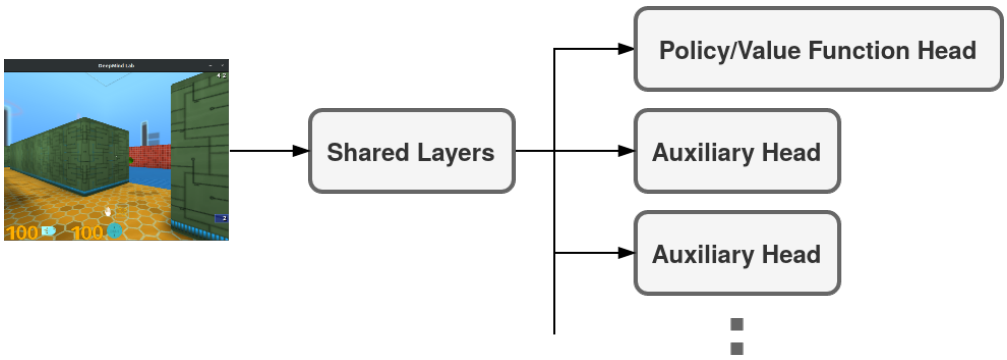


Figure 2-4: A schematic representation of an agent augmented with auxiliary tasks.

For demonstration purposes, let us consider the *UNREAL* framework proposed by [32]. UNREAL extends the base A3C loss (Equation 2-2), which consists of a policy loss term (actor), a value loss term (critic) and an additional entropy term to prevent premature convergence to suboptimal policies.

$$\mathcal{L}_{A3C}(\theta) = \lambda_\pi \mathcal{L}_\pi + \lambda_V \mathcal{L}_V - \lambda_H \mathbb{E}_{s\sim\pi}[H(\pi(x))] \tag{2-2}$$

The authors proposed to perform auxiliary reward prediction and pixel control tasks. The reward prediction task is formulated as a multi-class classification problem and is aimed at learning representations related to rewarding states. The pixel control task is a regression task which provides the agent with internal rewards based on differences in consecutive frames. This enriches the reward structure and stimulates the agent to explore. The resulting combined loss can be formulated as in Equation 2-3, where effects of each auxiliary loss is regulated by their respective weighting parameter. The loss function can be seperated to be applied directly from experience (*on-policy*), as for the base A3C loss, or based on replayed transitions (*off-policy*), as in the case of the pixel control loss.

$$\mathcal{L}_{UNREAL}(\theta) = \mathcal{L}_{A3C} + \lambda_{PC} \sum_c \mathcal{L}_{PC}^{(c)} + \lambda_{RP} \mathcal{L}_{RP} \tag{2-3}$$

Other auxiliary tasks proposed in the literature include depth and a loop closure prediction for navigation purposes [37], terminal prediction [38], and agent modelling in a multi agent setting [39].

2-6 IMPALA Architecture

Importance Weighted Actor-Learner Architecture (IMPALA) [40] is an off-policy, actor-critic architectecture that maintains a policy π and a value function V^{π}. The proposed architecture focuses on high throughput and multi task learning for improved generalization. Contrary to the popular A3C architecture [27], in which the actors share gradients with the learner, the IMPALA actors communicate experiences to the centralized learner(s) (Figure 2-5). This actor-learner decoupling allows actors to be independently distributed amongst different machines and asynchronously gather data for the training process. Consequently, the cheap aspects of simulation can be performed in parallel, while expensive computations can be passed on to a central GPU.

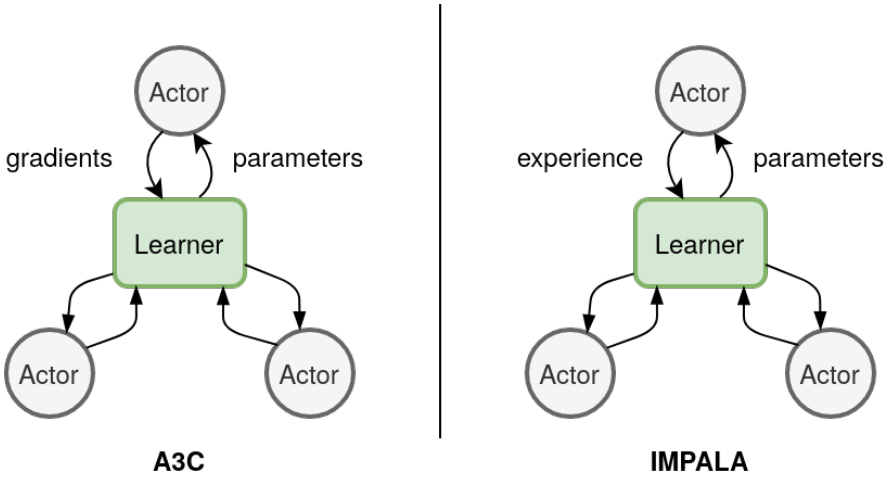


Figure 2-5: Comparison between the A3C architecture [27] and the IMPALA architecture [40]. The A3C actors compute gradients and share them with the learner while the IMPALA actors share observations with the learner, allowing the actors and the learner to be decoupled.

The algorithm is off-policy, hence the agent learns the *target policy* and value function based on data generated by the *behavior policy*. Each actor gathers trajectories of states, actions, rewards, as well as the policy distribution, based on the behavior policy. After a *roll-out* of n-steps, the trajectories are sent to the learner(s), which performs updates based on batches of trajectories from all actors. Native to off-policy methods, the behavior policy employed by the actors lags behind the policy that is being updated. In order to compensate for the lag and to prevent the learning from becoming unstable, a novel off-policy algorithm called *V-trace* was introduced by the authors. V-trace extends on the importance sampling corrections [41] by clipping the importance sampling ratios such that variance reduction can be achieved.

Formally, we seek to learn the value function V^{π} and the target policy π based on the trajectories $(x_t, a_t, r_t)_{t=s}^{t=s+n}$ generated by the behavior policy μ. Consider a parametric representation

of the value function V_θ and the current policy π_ω. The algorithm updates the value parameters θ and the policy parameters ω at training time s according to the following gradients:

$$(v_s - V_\theta(x_s))\nabla_\theta V_\theta(x_s) \tag{2-4}$$

$$\rho_s \nabla_\omega \log \pi_\omega(a_s|x_s)(r_s + \gamma v_{s+1} - V_\theta(x_s)) \tag{2-5}$$

where v_s represents the V-trace targets for the value approximations in x_s:

$$v_s := V(x_s) + \sum_{t=s}^{s+n-1} \gamma^{t-1} (\prod_{i=s}^{t-1} c_i) \delta_t V \tag{2-6}$$

Here, $\delta_t V := \rho_t(r_t + \gamma V(x_{t+1}) - V(x_t))$ represents the temporal difference for V, and ρ_t and c_i are the truncated importance sampling weights.

$$\rho_t := \min(\bar{\rho}, \frac{\pi(a_t)|x_t}{\mu(a_t|x_t)}) \qquad\qquad c_i := \min(\bar{c}, \frac{\pi(a_i|x_i)}{\mu(a_i|x_i)})$$

The truncation is assumed such that $\bar{\rho} \geq \bar{c}$. Furthermore, $\prod_{i=s}^{t-1} c_i = 1$ for $t = s$. Note that as $\pi = \mu$ (on-policy), the target reduces to the on-policy n-steps Bellman target and consequently V-trace reduces to the on-policy Bellman update. The truncated importance sampling weights each serves a different role. While $\bar{\rho}$ relates to the nature of the value function we converge to (the fixed point of the update rule), $\bar{c}$ impacts the speed of convergence to that (fixed-point) function and does not affect the fixed point itself. For untruncated ρ_t ($\bar{\rho} = \infty$), we get the target value function V^π. For finite $\bar{\rho}$ ($\bar{\rho} < \infty$), the fixed point lies between V^π and V^μ. Hence, the variance grows with $\bar{\rho}$, as the off-policy learning bias reduces for increasingly larger $\bar{\rho}$. Finally, an entropy bonus, the gradient of which is presented in Equation 2-7, is added similar to the A3C algorithm to prevent premature convergence.

$$-\nabla_\omega \sum_a \pi_\omega(a|x_s) \log \pi_\omega(a|x_s) \tag{2-7}$$

2-7 Conclusion

This chapter provided the foundation for the thesis work starting with the basics of reinforcement learning as a framework for sequential decision making problems. We discussed how deep learning techniques can be utilized to address the dimensionality and generalization issues of traditional reinforcement learning. The challenges of deep reinforcement learning were touched upon including credit assignment, learning stability and sample efficiency. One way to tackle these challenges is by incorporating the concept of auxiliary learning. Auxiliary learning is the utilization of subtasks to learn useful representations for the main task, that would otherwise have been learned much later in the process, if at all. Auxiliary learning is achieved by branching the model such that each additional head is responsible for its respective auxiliary task. The corresponding losses propagate back to the shared layers and contribute towards representation learning. State-of-the-art performances have been achieved by combining auxiliary learning with algorithms such as DQN [36] and A3C [32]. In the following chapters, we provide a formal framework for the exploration and coverage problem and propose to incorporate auxiliary learning with the more data efficient IMPALA actor-critic architecture and discuss how the setup can be trained to solve the exploration and coverage problem.

Chapter 3

Problem Formulation

In this chapter, we present a formal framework for the exploration and coverage problem. We consider a problem in which the agent is situated within an a priori unknown environment and it is required to search for areas of interest, also referred to as targets. The environment can be viewed as a 3D maze, in which the agent relies on RGB images, representing its frontal view, to manouvre. The agent does not have access to its position within the environment, nor does it have knowledge about the structure of the environment and the whereabouts of the targets. The environment is considered to be deterministic.

The objective of the agent is to manoeuvre within the environment and scan for randomly distributed (static) targets. When considering the example of a search and rescue mission, these targets can be viewed as hotspots with a high probability of actually finding survivors. The number of targets present in the environment is fixed and these targets do not respawn upon the arrival of the agent. Once the agent has managed to find all targets, the environment resets including its layout, the initial position of the agent and the position of the targets. The episode length is kept fixed and multiple environment resets can occur during an episode. The periods between resets are referred to as levels. The performance measure we employ is the cumulative reward (3-1) the agent receives each episode and consequently the number of targets it manages to find. Figure 3-1 provides an example representation of the timeline for the described problem.

$$R_t^\pi = \sum_{t=0}^{N} \gamma^t r_{t+1} \qquad \gamma \in [0,1] \tag{3-1}$$

The exploration and coverage task can be considered a POMDP as the agent does not have full information about the environment. The agent can only gain knowledge about the environment and its real states through the visual observations. As similar visual observations can be obtained from different states, e.g. when the agent traverses along a wall while facing it, the agent is required to make decisions under uncertainty. In other words, the projection from the observation space to the state space is non-unique. The environment remains unknown to the agent as it is randomized every level reset, as well as the initial state of the agent and

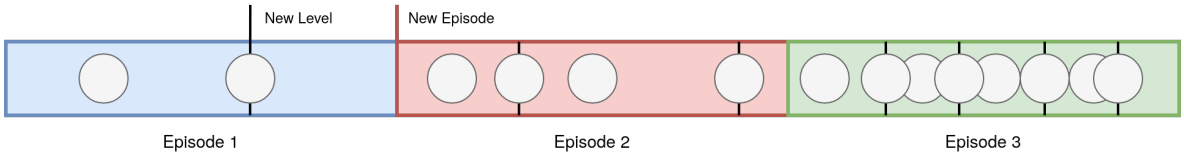


Figure 3-1: An example representation of the timeline of the exploration and coverage problem. The number of targets (gray circles) in each level, two in this example, and the episode length are fixed. Once the agent finds all the targets, a new level starts with the same number of targets. This process repeats until the end of the episode. An increased number of targets found in each episode suggests an improvement in performance.

the position of targets. The POMDP can be formally defined as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mathcal{O}, \mathcal{T}, \gamma \rangle$, where:

- $\mathcal{S}$ is the set of states $x \in \mathcal{S}$. The states contain information about the agent's experience up until time t and the whereabouts of the agent with respect to the environment;
- $\mathcal{A}$ is the finite set of actions $a \in \mathcal{A}$ available to the agent;
- $\mathcal{P}$ is the conditional transition function between states as a result of the action taken by the agent, formally formulated as $\mathcal{P}(x_{t+1}|x_t, a_t)$;
- $\mathcal{R}$ is the reward function describing the immediate rewards r the agent receives when taken an action in a certain state. Our reward strategy is to grant a reward for each target the agent manages to find;
- $\mathcal{O}$ is the set of observations $o \in \mathcal{O}$ consisting of raw images (RGB) representing the instanteous first-person view in the environment;
- $\mathcal{T}$ is the conditional observation transition probability function and is dependent on the state and the previously taken action, formally formulated as $\mathcal{T}(o_{t+1}|x_{t+1}, a_t)$;
- $\gamma \in [0, 1]$ is the discount factor weighting the importance of immediate rewards versus future rewards.

Each episode, the agent collects trajectories (sequences of observations, actions, and rewards) based on its current policy for a certain roll-out length. Batches from trajectories of all actors are sampled to train and update the model parameters. The latest policy update is utilized by the agent in the following roll-out.

In the above definition, coverage is not explitcity incorporated, but rather implicitly. The overlap between exploration and coverage lies in the fact that in order to find the targets, the agent needs to cover parts of the environment. The more and widely distributed the targets are in the environment, the more area (relatively) the agent needs to cover in order to find those targets. The number of rewards at the end of the episode will be determined by the speed at which the agent manages to find the targets. More details regarding the specific reward structure, the action set, the set of observations and the training procedure will be provided in 5.

Chapter 4

Methology

The formal setting for the exploration and coverage problem was provided in the previous chapter. This chapter underlines the principles of our proposed deep reinforcement learning approach to that formulation. We first provide the motivation behind our work, followed by a detailed analysis of the components of the proposed architecture and the optimization problem.

4-1 Motivation

Our work builds upon the IMPALA architecture by incorporating auxiliary learning. IMPALA was shown to outperform previous state-of-the-art algorithms such as A3C, and presented promising prospects in terms of multi-task learning and sample efficiency [40]. These aspects form crucial steps towards generative AI and hence we seek to build on this architecture for our approach. Specifically, we seek to explore whether the addition of auxiliary losses can improve the learning performance as it did to other DRL algorithms such as the A3C framework [27]. Our focus is to develop auxiliary tasks that can help the agent learn the exploration and coverage task.

Inspired by the Simultaneous Localization and Mapping problem and approaches, we propose a novel set of auxiliary tasks: *Pose Estimation* and *Local Map Prediction*. Intuitively, both tasks aim at building spatial knowledge about the environment. Similar to the SLAM problem, we hope that combined, these tasks would enhance each other and provide the agent with useful representations about the environment. The pose estimation task is specifically aimed at building information about the state of the agent with respect to the environment. The local map prediction task aims at building knowledge about the local proximity and its manouvrability. We believe both tasks can potentially be beneficial for future action selections. In contrast to SLAM approaches, in which one explicitly reconstructs the map and relies on accurate state estimations, we do not require the auxiliary losses to converge to zero as this is not the main goal. The main objective of the algorithm is still fundementally the search for the optimal policy and value function. The goal of the auxiliary tasks is to build

internal state representations that the agent may not have learned otherwise or would have learned much later in the process. Through shared network modules, the state representations learned from these tasks can influence the main policy and value function heads of the network, resulting in a more generalized policy. Mathematically, we hope the gradients from these auxiliary losses contribute towards faster convergence to the optimal policy and value function. In addition to the proposed pose estimation and local map prediction tasks, we also employ a modified version of the reward prediction task [32]. As the goal of the agent is to learn a policy that maximizes the cumulative reward, we believe trying to learn representations while relating them to rewarding states would benefit the learning process of the policy and value function.

Furthermore, we train the agent through a curriculum process. Humans and animals acquire complex skills by first learning the basics of a task and gradually integrate more and harder concepts into the learning process. By organizing the learning process in a structured and meaningful way, one can steadily gain the desired skills. In the context of machine learning, this is referred to as *curriculum learning* [42]. By teaching an agent to complete a relatively small and easy version of the task, the essential skills required to solve the actual task can be captured more easily in the neural network. The adaptation process then becomes less of a shock to the agent, when confronted with the actual task. We apply this concept in the hope of training the agent to get accustomed to various sizes of environments and various degrees of reward sparsity. In other words, we hope the agent learns to adapt to different environments.

4-2 Proposed Architecture

For the baseline, we utilize the deep residual model from the IMPALA paper, which originates from [43]. The residual structure allows more complex representations to be learned and was shown to outperform the shallow model in various settings [40].

We augment the base agent with the pose estimation and local map prediction tasks, as well as the reward prediction task. The extended agent will be referred to as *IMPALA-LM*, which stands for Localization and Mapping. In context of comparative studies presented in the following sections of this work, the base agent will be referred to as *IMPALA*. Figure 4-1 presents a schematic overview of IMPALA-LM in comparison to IMPALA. Here, we provide an overview of the architecture. For more details about the specific layers and activation functions in the network, the reader is referred to Appendix A.

The network has an *encoder-decoder* structure, in which the visual features from the input RGB image are encoded by a common residual convolutional unit. The encoded representation is the input to the different heads of the network: 1) the main LSTM unit for the approximation of the policy and value function, which in addition also takes the previous reward and action as inputs, 2) an additional LSTM block responsible for the estimation of the position and orientation of the agent, 3) the local map prediction module, and 4) the reward prediction module. This multi-head structure is common for auxiliary learning in DRL as well as multi task learning. It allows gradients from each task to propagate to the shared layers and contribute to the overall learning process.

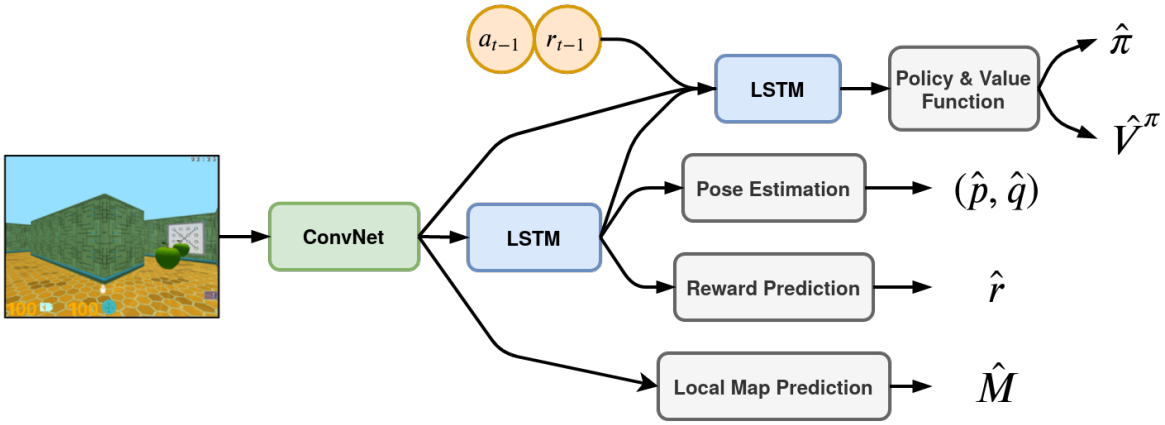


Figure 4-1: A schematic overview of IMPALA-LM, which extends the base IMPALA architecture with the Pose Estimation, Local Map Prediction and Reward Prediction auxiliary tasks.

4-2-1 Pose Estimation

The pose estimation task aims at building knowledge about locale specific features and learning about the actual states of the environment. We hope this information can be encapsulated and transferred to the main recurrent unit responsible for the policy and value function. Our pose estimation decoder receives the representations from the convolutional module and utilizes recurrency in the form of an LSTM unit. The output goes to two different heads, one for the estimation of the position and one for that of the orientation. The LSTM unit functions as an internal memory for the agent, allowing information to be stored about past observations and capturing encoded information about the states of the environment. Consequently, problems caused by the partial observability aspect of the environment are lessen to a certain degree.

While our experiments only involve planar motions in the xy-plane, we let the agent estimate its 3D position $\hat{p}_t$, representing the xyz-coordinate at each time step. Although the estimation of the z-component is unnecessary, the constant z-value helps convergence in the estimation task. We employ the mean squared error as the loss between the estimated 3D position and the actual position known to the designer. Due to the quadratic relation between the residual and the loss, and the fact that the position of the agent can take on a wide range of values, we normalize the target position by a factor hundred to prevent the loss from overpowering the optimization problem. This factor stems from the way the environment is constructed. More details can be found in Chapter 5.

With the orientation estimation head, we seek to estimate the planar orientation $\hat{q}$ (yaw/heading) of the agent. The yaw takes on a value in the range of (-180, 180) degrees. The orientation prediction task is defined as a multi-class classification problem, where the 360 degrees range is divided into 12 bins of 30 degrees. Formulating the problem as a classification task simplifies the problem and makes learning the task easier in comparison to a regression formulation. The cross entropy is used to compare the network output with the one-hot encoded target.

The resulting pose estimation loss $\mathcal{L}_{pose}$ can be formulated as in (4-1).

$$\mathcal{L}_{pose} = \lambda_p \frac{1}{N} \sum_{t}^{N} \Big(p_t - \hat{p}_t \Big)^2 + \lambda_q \Big(- \frac{1}{N} \sum_{t}^{N} q_t \log(\hat{q}_t) \Big) \tag{4-1}$$

where λ_p and λ_q are the weighting parameters of the position and orientation losses respectively, and N represents the number of training samples. The squared residual represents total error of each sample.

Drift is one of the key problems in SLAM. Drift occurs as the robot tracks its position relative to landmarks in the environment. Since the pose is estimated relative to the landmarks and each landmark position possesses a measurement error, the estimation error accumulates with time and movement of the robot. The error can be drastically reduced as the robot returns to a previous pose and detects the same landmarks, known as loop closure, and as the robot gains information about the uncertainty of its measurements. While our estimates are absolute, we seek to constrain the pose estimates with a similar intension. Specifically, we minimize residual between the difference in consecutive position estimates $\hat{p}_t$ and $\hat{p}_{t-1}$, and the difference in the actual consecutive positions. We call this the consistent position estimation loss $\mathcal{L}_{ce}$, which is defined as the following mean squared loss:

$$\mathcal{L}_{ce} = \lambda_{ce} \frac{1}{N} \sum_{t}^{N} \Big((p_t - p_{t-1}) - (\hat{p}_t - \hat{p}_{t-1}) \Big)^2 \tag{4-2}$$

4-2-2 Local Map Prediction

The local map prediction task is similar to the pose estimation task in the sense that it aims at building more knowledge about the locale specific features of the environment. We hope the agent is able to find the relation between the visual inputs and the corresponding local proximity. As the local proximity provides information about the free and occupied space around the agent, this information can potentially have a positive effect on future action selections and hence contributes to the learning of the policy.

The local map prediction network is inspired by a submodule proposed in [44]. Contrary to [44], in which additional observations are used by the agent, our network only depends on visual observations. The network takes the encoded features from the convolutional unit and decode the information to form a reconstruction of the local map excerpt. The local map excerpt is a tensor representing an image of the local map structure (Figure 4-2). The estimated pose is given as an additional input to the network. We apply a sigmoid activation in the final layer to output the belief of the agent about the structure of the local map, i.e. whether certain parts the local map excerpt is free or obstructed. We consider a grayscale excerpt, where obstructed and free areas are encoded with ones and zeros respectively. The target is the actual local map M, which is contructed dependent on the actual position of the agent.

We utilize the mean squared error as the local map prediction loss on the residual between the estimated local map $\hat{M}$ and the target local map M:

$$\mathcal{L}_M = \lambda_M \frac{1}{N} \sum_{t}^{N} \Big(M - \hat{M} \Big)^2 \tag{4-3}$$

Here, the squared residual denotes the total, element-wise squared difference over all pixels between the predicted local map excerpt and the target excerpt.

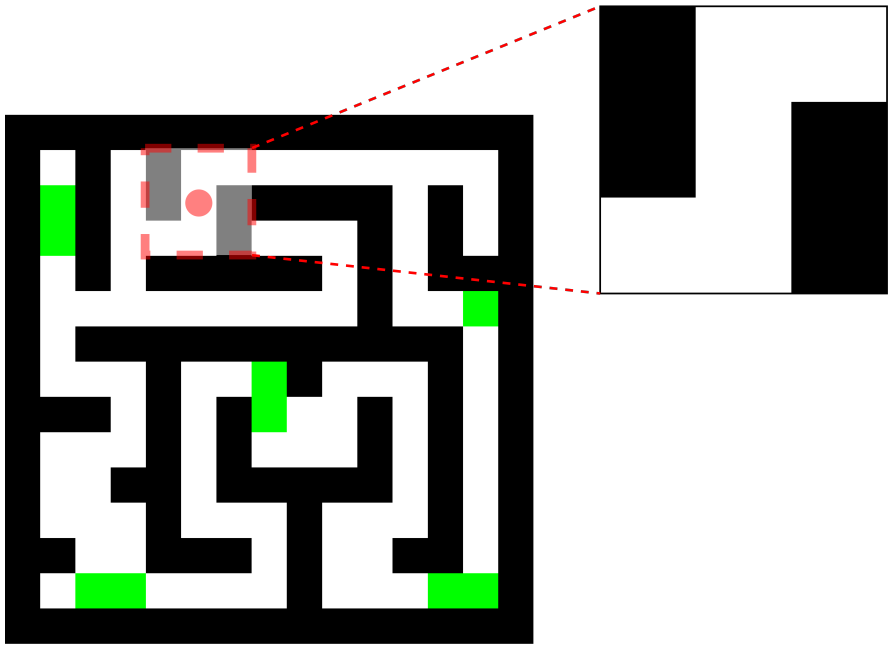


Figure 4-2: A representation of the layout of the environment. The local map excerpt contains information about the true local proximity of the agent and is utilized for the local map prediction loss.

4-2-3 Reward Prediction

The reward prediction task has been previously proposed in [32]. The agent outputs its belief about whether the subsequent observation would provide a negative, zero, or positive reward, based on the past three observations. Training is performed through samples from a replay buffer in which zero and non-zero rewards are equally represented. The idea is that the reward prediction task resembles value learning focusing on immediate reward ($\gamma = 0$) without biasing the policy.

Throughout this work, training is performed off-policy and we do not utilize an experience replay buffer. Hence, rather than utilizing consecutive observations from a replay buffer to form historical context, we opt for a recurrent unit. In fact, the reward prediction head makes use of the same LSTM unit as the pose estimation head. As our reward strategy only involves positive rewards for finding targets, we use a binary classification across two classes: zero and nonzero (positive). However, as this loss is relatively easy to optimize, and due to reward sparsity, zero rewards are overrepresented, we utilize a softmax nonlinearity as the final layer rather than a sigmoid nonlinearity. We hope this measure prevents the network from becoming overconfident in predicting zero rewards. The reward prediction loss can be formulated as the following cross entropy:

$$\mathcal{L}_r = \lambda_r \Big(-\frac{1}{N} \sum_{t}^{N} r_t \log(\hat{r}_t) \Big) \tag{4-4}$$

4-3 Learning Objectives

With the proposed auxiliary losses introduced, we summarize the learning objectives of the proposed deep reinforcement learning algorithm. At the foundation, IMPALA maintains a policy and value function, which are parameterized by θ and trained through an actor-critic update in order to maximize the cumulative reward given in (3-1).

The updates are performed with entropy regularization based on the gradients presented in (2-4), (2-5) and (2-7), following the base loss function:

$$\mathcal{L}_{IMPALA}(\theta) = \mathcal{L}_V + \mathcal{L}_\pi + \mathcal{L}_H \tag{4-5}$$

$$\begin{aligned}\mathcal{L}_{IMPALA}(\theta) = \lambda_V \frac{1}{2}\Big(v_s - V_\theta(x_s)\Big)^2 \\ + \lambda_\pi \rho_s \log\pi_\theta(a_s|x_s)\Big(r_s + \gamma v_{s+1} - V_\theta(x_s)\Big) \\ - \lambda_e \sum_a \pi_\theta(a|x_s)\log\pi_\theta(a|x_s)\end{aligned} \tag{4-6}$$

where v_s is the V-trace target (2-6).

The extended algorithm, IMPALA-LM optimizes the combined loss function (4-7) with respect to the joint parameters θ. The pose estimation, consistent position estimation, local map prediction, and reward prediction losses $\mathcal{L}_{pose}$, $\mathcal{L}_{ce}$, $\mathcal{L}_M$, and $\mathcal{L}_r$ respectively, have been presented in this chapter.

$$\mathcal{L}_{IMPALA-LM}(\theta) = \mathcal{L}_{IMPALA} + \mathcal{L}_{pose} + \mathcal{L}_{ce} + \mathcal{L}_M + \mathcal{L}_r \tag{4-7}$$

All losses are performed based on batches of experiences from all actors. These experiences are gathered for a certain unroll length (roll-out) based on the behavior policy employed by the agents.

The auxiliary losses introduce a number of weighting terms as additional hyperparameters. To mitigate the drawback of manually searching for a high performing combination of weighting parameters, we investigate the concept of learnable parameters presented in [45]. An arbitrary loss function with auxiliary hyperparameter λ (4-8) can be substituted by a formulation in which α and β are the learnable parameters (4-9). This formulation is related to homoscedastic uncertainty, and scales the loss terms in a probabilistic manner. Here, the learnable parameter represents the logarithm of the variance of the respective task $\beta := \log\sigma^2_{aux}$. When the variance is large, the residual term $e^{(\cdot)}$ regularizes to temper the loss contribution. An additional regularization term $+(\cdot)$ is used to prevent the network from predicting infinite uncertainty (zero loss). These parameters are learned through backpropagation with respect to the loss function.

$$\mathcal{L} = \mathcal{L}_{base} + \lambda\mathcal{L}_{aux} \tag{4-8}$$

$$\mathcal{L} = \mathcal{L}_{base}e^{-\alpha} + \alpha + \mathcal{L}_{aux}e^{-\beta} + \beta \tag{4-9}$$

4-4 Conclusion

In this chapter, we provided the details to the proposed deep reinforcement learning algorithm IMPALA-LM. Our work builds on the IMPALA actor-critic agent by incorporating a novel

set of auxiliary tasks, namely *Pose Estimation* and *Local Map Prediction*. Additionally, we enforce consecutive position estimates to be close to the actual relative displacement with a *consistency loss* and utilize a modified version of the reward prediction task as proposed in the literature. The agent is trained through a curriculum process, in the hope of gradually building up the skills for the exploration and coverage task. In the following chapter, we outline the evaluation process of IMPALA-LM and present the accompanying results.

Chapter 5

Experiments

In this chapter, we describe the evaluation process of the proposed method and present the findings of our project. A description of the engine in which we perform our evaluations will be provided, followed by the details of the implementations. We present the results of the experiments to evaluate the performance of IMPALA-LM compared to the base IMPALA agent. We investigate the effects of learning auxiliary weighting terms and assess the performance of our agent compared to a human controlled agent.

5-1 Experiment Setup

Various environments have been utilized in previous works to evaluate visual deep reinforcement learning such as VizDoom [46], Project Malmö [47] or Airsim [48]. In this work, we make use of the DeepMind Lab environment [49]. Deepmind Lab is a 3D first-person game platform designed for AI agent research. Similar to Doom and Malmö, the Deepmind Lab environment involves scenes that are synthetic. The lack of realistic observations allows the environment to be relatively simple. Consequently fast rendering speeds can be obtained allowing shorter training times of AI agents, which is especially important for DRL as methods are vastly sample inefficient. Other main benefits of the Deepmind Lab environment are its customizability and extendability. While the observations are not realistic, we hope that, with the rich variety, the trained agent can learn complex behaviors that are transferable to other environments where the visuals are significantly different. Additionally, the Deepmind Lab environment has a large supporting community and many researches in DRL have been benchmarked using this environment [37], [32], [50].

In order to accomodate the desired training procedure, we modified one of the contributed environments from the DMLab-30 set, called `explore_object_locations`. DMLab-30 is a set of environments consisting of a large variety of different problems aimed at creating a generalized agent through multi-task learning. The `explore_object_locations` environment serves our intended problem of searching for areas of interest. In each level of the environment, the agent is placed within a randomized maze. It is required to collect targets, represented

by apples. Upon reaching the target, defined as the relative distance between the agent and the target, the agent receives a positive reward $r = 1$. The targets do not respawn upon the arrival of the agent and the number of targets in each level is fixed. When all targets have been found by the agent, the level resets. After each reset, the layout of the environment is randomized, as well as the initial position of the agent and the positions of the targets. As the episode length is also fixed, multiple level resets can occur during an episode. The measure for evaluating the training performance is the cumulative reward, also referred to as *return* (3-1). An increase in the cumulative reward as time progresses suggests the effectiveness of the learning process. Figure 5-1 shows possible input images and the variation in visual scenes in the environment.

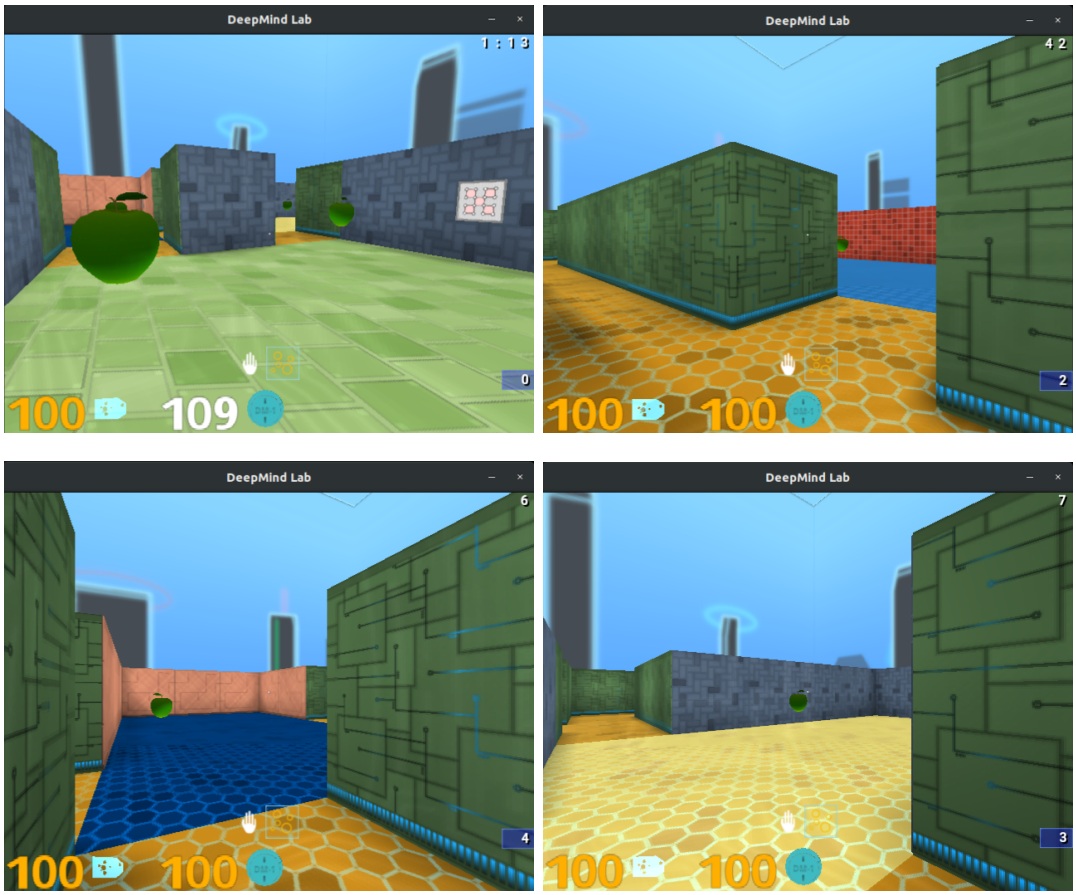


Figure 5-1: The agent is subject to a variety of visual observations from the environment. Its goal is to find targets represented by apples.

We investigate the performance of our proposed approach under various settings. In terms of the training performance, with the cumulative reward as performance measure, we compare IMPALA-LM and the contribution of the submodules with the base IMPALA agent. We also go into the effects learning the auxiliary weighting parameters on the training performance in comparison to the setting with equally weighted, fixed auxiliary scalings. In the fixed weighting setting, the auxiliary weights are set to the natural first guess of $\lambda_i = 1.0$. For comparative studies on the learning performance, we only consider Stage 1 of the learning

process as presented in Table 5-1. For the final agent, which is used for inference, we apply curriculum learning by gradually increasing the difficulty of the exploration and coverage task through multiple stages. We consider the size of the environment, the density of rewards, and the variations in visuals to be the contributing factors to the level of difficulty. The variation in visuals refers to the different textures or themes in the environment (Figure 5-1). When the variation equals one, the walls, floor and hallways have a similar look throughout the environment. The details to each stage of the learning process is provided in Table 5-1. Through the multi stage training procedure, we hope the agent is able to adapt to various sizes of search territories, and with that sparser reward signals, as well as to variations in visual scenes. With regards to the inference performance, we compare the IMPALA-LM agent, trained through curriculum, with a human controlled agent. The human has access to a similar action set as the trained agent. We hope this provides more meaning to the training performance return value and reflects the effectiveness of the autonomous agent at the exploration and coverage task.

Table 5-1: Stages of training. The map size follows ASCII encoding, hence a size of $w \times h$ means the position of the agent can range from 0 to $100w$ and $100h$ respectively.

Stage	Map size	Number of targets	Variation	Training duration (frames)
1	11 x 11	8	1	10^7
2	11 x 11	8	1	10^7
3	15 x 15	8	1	20^7
4	19 x 19	10	5	20^7

5-2 Implementation Details

Similar to the original work on IMPALA [40], we employ first-person observations of 96 by 72 pixels (Table. 5-2). At each time step, the agent receives the instantaneous visual observation from the environment. In addition to the input image, the agent also makes use of the last action a_{t-1} and the last received reward r_{t-1} as inputs. The true position and orientation of the agent, as well as the layout of the environment are accessible to the designer. Based on this information, we construct the target local map excerpt of 20×20 pixels, corresponding to the local proximity of the agent. The data is used in batches of experiences (from all actors) for the evaluation of the losses.

Table 5-2: Environment parameters for the input image and the local map excerpt in pixels.

Parameter	Value
Input image width	96
Input image height	72
Local map width	20
Local map height	20

The employed action space (Table 5-3) consists of six different discrete actions: *forward*, *backward*, *move left*, *move right*, *turn left*, *turn right*. The human has access to the same

action set in our experiments. The respective encoding is processed internally based on a frame rate of 60 frames per second and results in the corresponding new states and reward. The encoding for the turning actions represent the unit pixels. With the current settings, the agent turns approximately 8 degrees per frame.

Table 5-3: Action set used in all experiments and the corresponding Deepmind Lab encoding.

Action	**Deepmind Lab encoding**						
Forward	0	0	0	1	0	0	0
Backward	0	0	0	−1	0	0	0
Move left	0	0	−1	0	0	0	0
Move right	0	0	1	0	0	0	0
Turn left	−20	0	0	0	0	0	0
Turn right	20	0	0	0	0	0	0

Table 5-4 presents the fixed parameters used for all experiments and for both the base agent and the extended agents. Note that we do not tune these hyperparameters for IMPALA-LM, but rather utilize the parameters presented in [40]. As this set of hyperparameters has been tuned extensively for the base agent, we believe using the same set of parameters provides a fair comparison. The performance difference would be the result of the proposed auxiliary losses rather than of a better tuned set of hyperparameters. However, it is to be noted that the hyperparameters have been tuned to perform across the board in various different tasks rather than specifically for our problem, and with the addition of the auxiliary losses, a different set of hyperparameters could be a better fit for IMPALA-LM.

Table 5-4: Fixed parameters used in all experiments.

Parameter	**Symbol**	**Value**
Training		
Discount factor		0.99
Batch size		16
Unroll length		100
Number of actors		20
Action repetitions		4
Loss		
Policy loss scaling	λ_π	1.0
Value function loss scaling	λ_V	0.5
Entropy loss scaling	λ_H	0.00025
Optimizer		
Learning rate		0.00048
RMSProp decay factor		0.99
RMSProp epsilon regularization		0.1
RMSProp momentum		0.0

Given the available computational resources, we utilize a single machine setup rather than the distributed setup. With 20 actors, an unroll length of 100 steps, and an action repeat of 4, we reach a throughput in the lower range of 2500 to 3000 frames per second with IMPALA-LM. The base IMPALA setup results in a slightly higher throughput, on average around 3000 frames per second. This demonstrates that the computational expense of the extended agent is in the same order as the base agent.

5-3 Adaptive Auxiliary Weights

With the introduction of the auxiliary losses, additional hyperparameters are introduced on top of the standard RL hyperparameters. Hyperparameter tuning can be a highly extensive process and it is computionally expensive, even if all parameters are set similar to the base agent. As new hyperparameters are introduced to the problem, and as changes are made to the agents network, different combinations of hyperparameters might yield better results. To mitigate the extensive hyperparameter tuning process, we investigate the effects of learning auxiliary parameters [45]. Note that only the proposed pose estimation and local map prediction losses are considered in this experiment. The fixed weights are set equal to one for both auxiliary tasks and are therefore equally weighted as the policy gradient loss. We perform the training for multiple seeds and present the average return over all seeds. The individual results can be found in Appendix B.

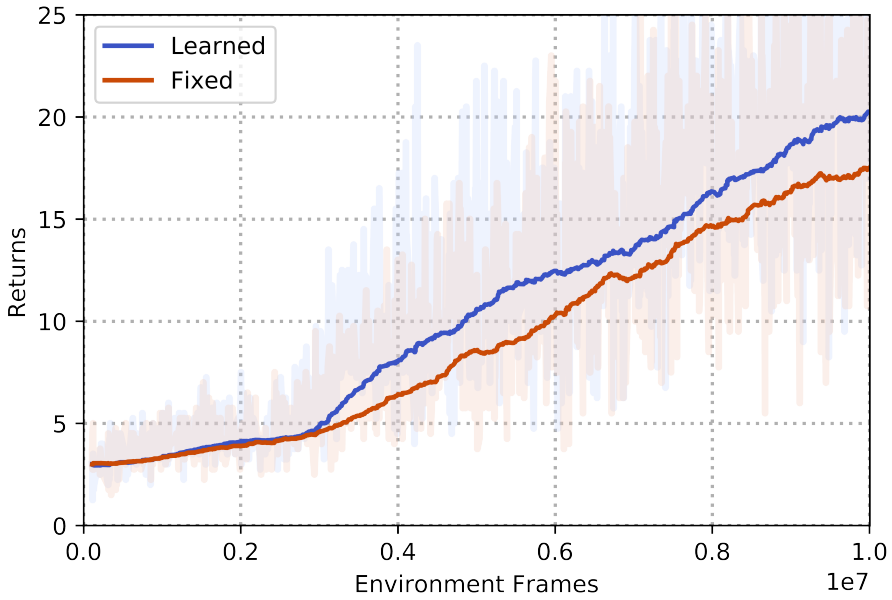


Figure 5-2: A comparison of the learning curve between an agent that learns the auxiliary weights, for the pose estimation and the local map prediction tasks, and an agent with fixed auxiliary weights. In the latter, the weights are set to $\lambda_p = \lambda_q = \lambda_M = 1$ and are equal to the weighting term of the policy gradient loss.

It can be observed that the agent that learns the auxiliary weights slightly outperforms the agent with fixed auxiliary weights. This result suggests the effectiveness of learnable auxiliary hyperparameters. Consequently, further evaluations will be performed using auxiliary weights

learned by the agent. However, as the comparison is performed with respect to one set of auxiliary weights, we cannot conclude that this method produces the best possible results for the extended agent. It is to be considered a tool to alleviate the burden of searching for a well performing set of parameters and the need to repeat the process when new auxiliary losses are introduced or changes are made to the network architecture. It functions as a regulator for the contribution of each term to the overall optimization problem. Since it is difficult to evaluate the effects of the gradient from each respective auxiliary loss term, we believe this is beneficial.

5-4 Baseline Comparison

In this section, we demonstrate the effectiveness of the proposed auxiliary losses and compare the training performance of IMPALA-LM to the base IMPALA agent. The weighting parameters for all the auxiliary losses are learned by the agent. The following figure presents the average result over multiple seeds.

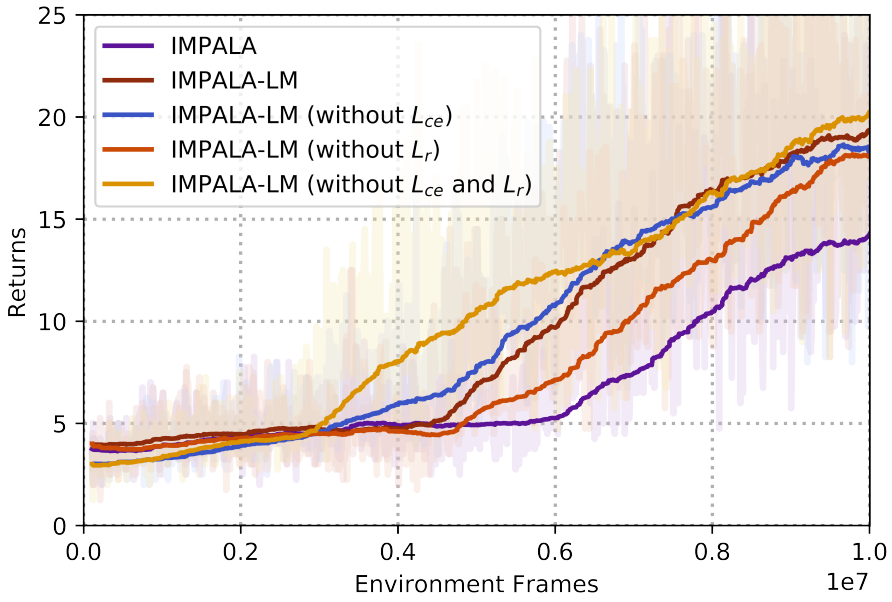


Figure 5-3: A comparison between IMPALA-LM, with different combinations of auxiliary losses, and the base IMPALA agent. The auxiliary weights are learned by the agent.

The learning curves demonstrate that IMPALA-LM outperforms the base agent. For example, with the auxiliary pose estimation and local map prediction losses (yellow curve), the agent requires almost 40% fewer number of samples in comparison to the base agent to reach an average return of 8, which is the total number of target in each level.

While all versions of IMPALA-LM outperform the base IMPALA agent, we can observe that the addition of the reward prediction loss and the consistency loss does not enhance the training performance. If anything, the addition of these losses has a diminishing effect. There are multiple potential contributing factors to this phenomenon. First, the additional gradients do not contribute towards driving the optimization problem to its (global) optimum. While

the intuition is to build useful representations with auxiliary tasks, it is not trivial to know when a task contributes positively to the optimization problem. The additional gradients might have a negative effect on the convergence of the policy. On the same note, the auxiliary tasks may be too difficult for the agent to learn, resulting in highly variable gradients from training step to training step. While the auxiliary weights are learned, this learning process does not directly involve the relative gradients, but it is rather the result of the total loss. Works such as [51] proposed measures to tune the auxiliary weights based on the relative gradients. In [51], the cosine similarity between the gradients is used as adaptive auxiliary weights. Second, the auxiliary tasks do not capture useful internal state representations as hoped. We have experimented with more complex architectures in the hope that more information can be encapsulated in the network, as well as regularization measures such as dropouts to prevent overfitting. However these experiments had varying degrees of success and provided no significant improvements. Hence, the current network structure might be at cause. Third, adding extra loss terms increases the complexity of the optimization problem. The increase in complexity may be to the extent that it outweights the positive effects of the additional gradients. This can potentially cause the agent to not learn anything more often than without the additional heads and loss terms. Furthermore, as we are dealing with moving targets rather than fixed targets as in supervised learning, and the optimization problem is highly dependent on the data distribution, the average over four different seeds might not be enough to paint an accurate picture.

5-5 Curriculum Learning

Through a curriculum process, we seek to increasingly build the skills required for the agent to successfully accomplish the exploration and coverage task. The different stages of curriculum are provided in Table 5-1. Figure 5-4 presents the results to the curriculum learning process. As the previous section has shown, the reward prediction and consistency losses do not have a significant effect on the learning performance, hence this experiment only involves the pose estimation and local map prediction task.

Stages 1 and 2 are similar, hence, as the performance increases throughout Stage 2, the agent has not yet reached the steady-state in terms of performance. In stages 3 and 4, we see a significant dropoff in returns as the sparsity of reward signals increases while the episode duration stays unchanged. While Stage 3 shows a recovering agent that learns perform better over time, Stage 4 shows a steady learning curve. A dropoff in returns is to be expected as the density of targets in the environment is decreased. We expect training the agent for a longer duration in the earlier stages, such that the exploration skills improves for those environments, will better on the learning performance in the latter stages. The danger of increased sparsity is that as the timespan between reward signals significantly decreases, the agent might largely deviate from a previously well performing policy and end up *delearning*.

5-6 Human Level Comparison

In order to gain more insights on the returns achieved by the agent presented in previous sections, we analyze the performance of the trained agent against a human controller. We

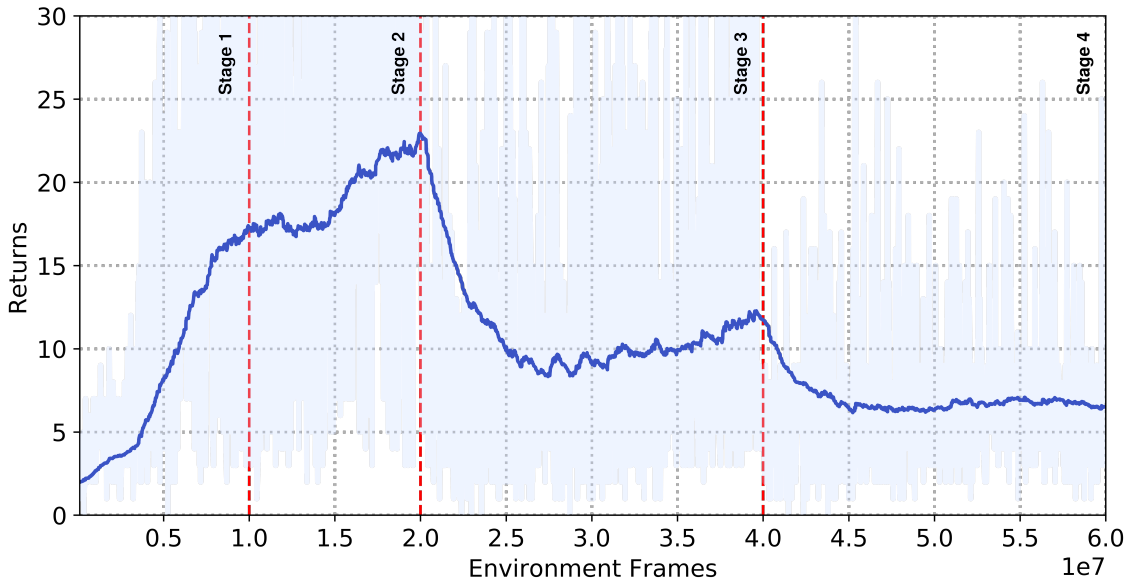


Figure 5-4: The learning curve of the curriculum learning process.

conduct experiments in which the human has access to the same six-tuple action set as the agent. The experiments are performed in three different environments of different complexities as presented in Table 5-5. Test 1 and Test 2 are taken in the same environment as Stage 1 and Stage 4 of the curriculum learning process respectively. Test 3 lies in between the two in terms of complexity. Note that the reward density in Test 3 is on a similar scale compared to Test 1 (240% increase in area and 225% increase in the number of rewards).

The results in Figure 5-5 show the distribution of the inference score or return over ten different runs. In the case of the trained agent, the score of each run represents the mean over 10 episodes.

Table 5-5: The settings to the inference experiments.

Test	Map size	Number of targets	Episode duration (seconds)
1	11 x 11	8	90
2	19 x 19	10	90
3	17 x 17	18	120

In Test 1, we can observe that the agent slightly outperforms the human controller. Interestingly, Test 3 shows a similar phenomenon. We believe by having trained the agent on various different layouts, subject to variations in visual observations, and with varying degrees of reward sparsity, the agent is able to adapt to this new environment. Eventhough the size of the environment is larger, the density of rewards is approximately similar in these two tests. In Test 2, where the agent was previously shown to struggle with the low density of reward signals, we see that the performance of IMPALA-LM is inferior to the human controlled agent. In Appendix B, we present additional results on the inference performance of the trained agent by plotting the trajectory for examples of each of the three environments.

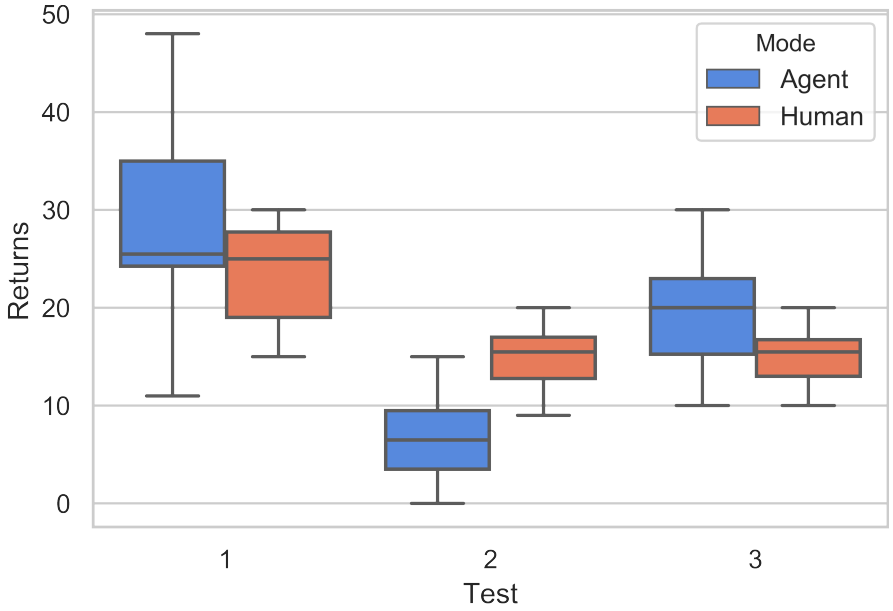


Figure 5-5: A comparison between the inference performance of an agent, trained through the described curriculum, and of a human controlled agent with a similar action set.

Our observation is that the agent is highly effective in gathering targets when the density is relatively high and when targets are present in packs, in other words distributed in groups of several targets. Possibly, it is because the agent is more direct when targets are observed, while a person might be less efficient in controlling the movement towards the target. This suggests that the agent has learned the essential skills of recognizing the targets and controlling its movement towards them.

However, the agent seems to lack an effective exploratory strategy to search for vastly distributed targets. When the targets are very sparse and highly distributed across the environment, it is increasingly important to remember all the routes one has traversed to minimize the time spent going over the same paths. Moreover, we noticed that high scoring runs in these environments involve a slightly different set of skills. A direct approach involving mostly translations is less effective than strategies in which one scans an open space through turns and manouvres to the subsequent open spaces to repeat the process. We believe these high level skills is where the agent has missed the mark. A potential contributing factor could be that the agent has been biased towards a direct manouvre strategy through the training in denser signalled environments. In addition, not enough historical information may have been captured through the recurrent unit to effectively search for the unexplored areas. Using recurrency on different timescale might have a possitive effect as the agent needs the shortterm vision to translate locale specific information to an appropriate action, but also the longterm memory to recognize previously traversed paths.

Furthermore, an important distinction between IMPALA-LM and a human controlled agent is that the human is much more consistent. The agent might reach a very high score one time, but underperforms on another. This is also visible from the variance in the learning curves presented in previous sections. While it is inherent to RL and its trial and error nature, this phenomenon can also be what is referred to as *catastrophic forgetting* [52] in the literature.

When presented with new data, the agent might abruptly forget previously well performing policies.

5-7 Conclusion

This chapter presented the setup to evaluate the learning performance of the proposed approach, as well as its inference performance. We discussed the different aspects to learnable auxiliary parameters and showed that an agent learning the weighting terms in addition to the prediction tasks outperforms an agent with a fixed set of auxiliary weights. IMPALA-LM was shown to outperform the base IMPALA agent, suggesting the positive contributions of our proposed auxiliary tasks. However, with the addition of a loss contraining consecutive position estimates and a reward prediction loss, the learning performance is not improved. We discussed several potential contributing factors to this observation. IMPALA-LM, trained through a curriculum process, was shown to outperform a human controller in two out of three scenarios. However, the reoccurring phenomenon is that the agent achieves a high performance when the targets are relatively dense, but struggles when the time between reward signals is much larger than previously experienced.

Chapter 6

Conclusions and Recommendations

This chapter summarizes the findings of this thesis. Moreover, it touches on the remaining challenges and provides an outlook for future works.

6-1 Summary

Exploration and coverage refers to the search and gathering of important information in unknown environments. This work specifically focuses on finding areas of interest, also referred to as targets, in such environments with an autonomous agent. Autonomous agents possess the potential to be more efficient and effective than humans and allow humans to be pulled out of the equation when lives may be at risk.

While traditional methods often require various forms of active range sensors or can swiftly rack up in hardware complexity, the development in computer vision techniques and artificial intelligence offers promising prospects of deploying single camera systems without loosing the exploratory capabilities. Particularly, our work focuses on reinforcement learning techniques. Reinforcement learning is a sequential decision making process where the agent learns to solve complex tasks through trial and error, a process very similar to the way humans and animals learn. By shaping the external feedback from the environment, one can direct the learning process in order to teach the agent solve a certain problem. With the recent advances in deep learning, the integration of deep neural networks as function approximators in reinforcement learning paves the way to tackle high-dimensional control problems without the requirement of modelling the system. However, deep reinforcement learning, and its applicability in the real world, is limited by the excessive need for vast amounts of interactions before the agent is able learn an appropriate behavior. In this thesis, we explore the technique of *auxiliary learning* to enhance the learning process. By introducing relatively easy subtasks, we hope to build useful internal representations that the agent would not have learned otherwise, or would have learned much later in the process. Specifically, we introduce the *Pose Estimation* and *Local Map Prediction* tasks to build locale specific knowledge and relate the visual observations

to the structure of the local proximity. We name the proposed agent IMPALA-LM, which extends the state-of-the-art IMPALA algorithm.

Our experiments show that by learning to estimate its pose and local structure concurrently to finding the optimal policy, the agent manages to reach certain levels of performance earlier in the learning process compared to the base IMPALA agent. In particular, IMPALA-LM learns to find all the targets with almost 40% fewer training samples on average than IMPALA, in our baseline comparison. We show that finding appropriate implementations of auxiliary tasks and a well performing set of hyperparameters can be challenging, as the contributions of the auxiliary losses to the main problem are not trivial. This is apparent as the additional consistency loss and reward prediction loss did not have a positive effect on the learning performance. A balance between the potential sample efficiency benefits and the increase in complexity of the optimization problem should be found. The IMPALA-LM agent, trained through a curriculum process, marginally outperforms a human controlled agent in two out of three tested scenarios. In the third scenario, when confronted with increasingly sparse signalled environments, the agent seems to lack the exploratory capabilities to search for all obstructed targets. This demonstrates that eventhough IMPALA-LM outperforms IMPALA in terms of the learning performance, and the learned exploratory behavior works for relatively simple cases, enhancements are necessary in order to generalize this behavior to more difficult environments. We believe that in essence, the agent has learned to recognize the targets and effectively manouvre towards them in the early stages of the curriculum process. This direct behavior is beneficial when the number of targets is relatively high, as the likelihood of observing a target is relatively. In the scenario that targets are highly distributed and sparse, we believe a more observant, high level behavior is required, which the agent has failed to learn in the latter stages of curriculum. This failure may be caused by the curriculum process itself, as it can bias the agent too much towards a certain behavior.

6-2 Recommendations

Auxiliary learning presents an interesting perspective on the sample efficiency problem of deep reinforcement learning techniques. It overlaps the subfield of multi-task learning and offers insights to the transfer of knowledge from one (sub)task to another. This thesis only presents a small contribution in terms of a novel set of auxiliary tasks for the exploration and coverage problem, and potentially for other mobile robotics problems. The major challenges of auxiliary learning lie in the design of appropriate tasks, finding a well performing combination of tasks, and tuning the respective weighting parameters. Throughout this thesis, we have mentioned several works that use the relative gradients as a measure for the effectiveness of auxiliary losses [53], [51]. We believe the research into understanding the actual contribution (of combinations) of auxiliary losses and adapting the weighting parameters accordingly is of high importance. The design of new auxiliary tasks that can potentially be beneficial for reinforcement learning problems in general rather than task specific, deserves equal attention.

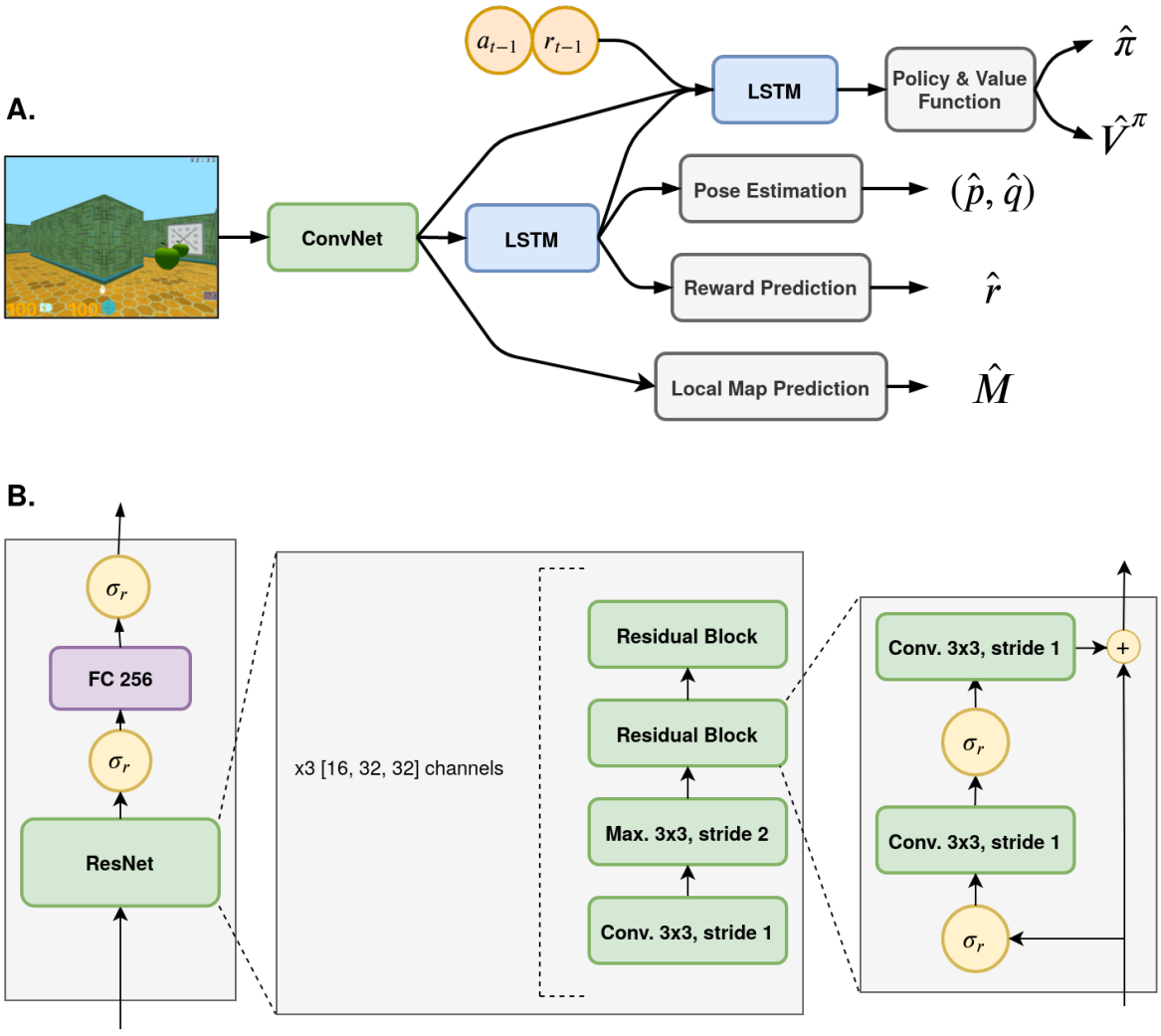
With regards to our implementation, we are interested in whether training the agent such that convergence in performance is reached in the early stages of curriculum will have more positive effects when transferring to more challenging environments. As the ultimate goal is an agent that can generalize across different environments, training the agent in other environments, including those with more realistic scenes, environments with outdoor scenes,

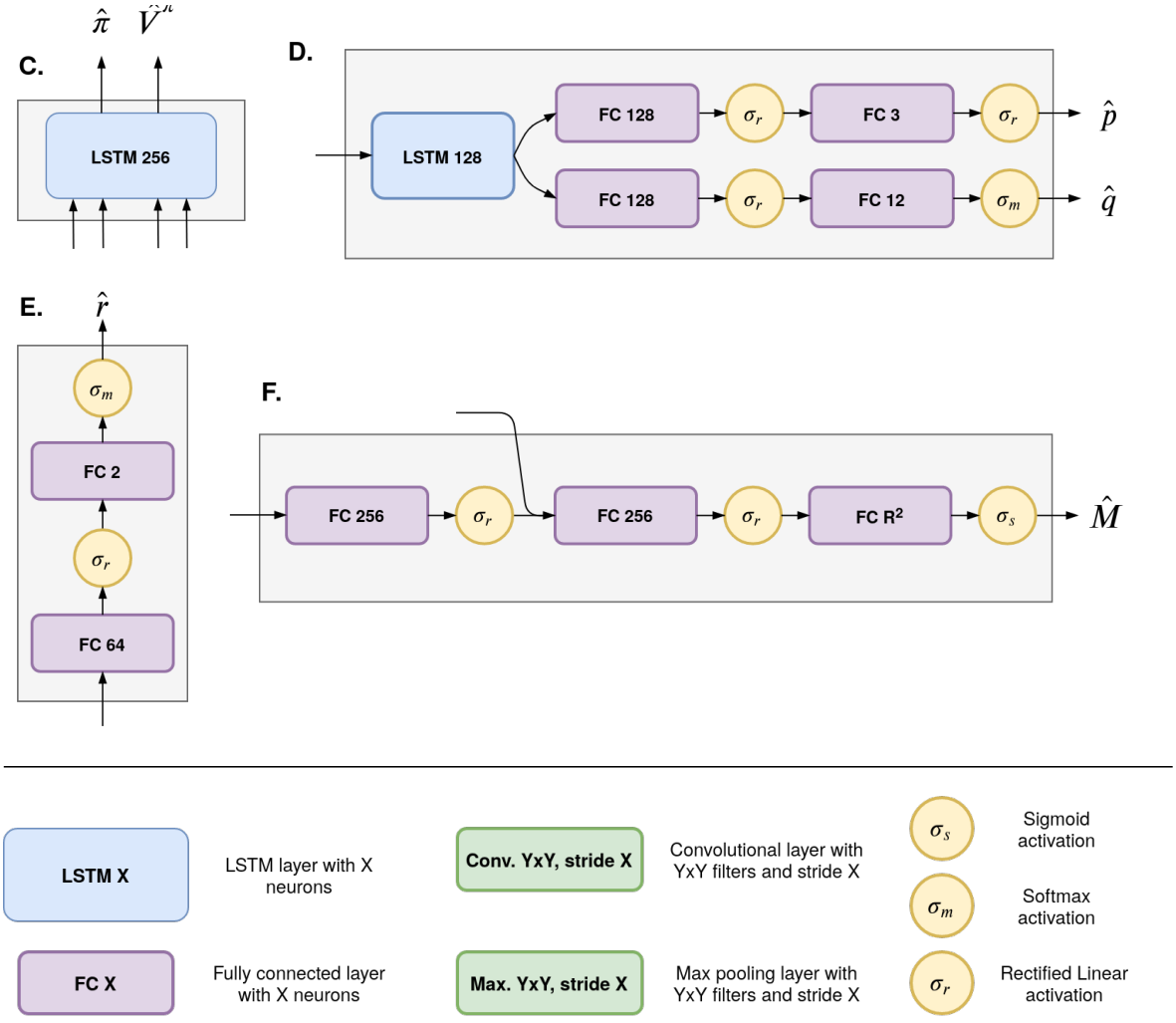
or those with various degrees of luminosity, forms an important aspect. By increasing the variety in visuals, the adaptivity of the agent can be enhanced. Furthermore, the integration of multi timescale recurrency is be interesting, as the agent can benefit from various forms of abstractions acquired on both shortterm as well as longterm.

The results in this work have been presented with the intension of portraying the contribution of the proposed auxiliary tasks from the start of training. An interesting research direction is to investigate the effects of integrating (pre-trained) auxiliary tasks in different stages of training. On the same note, the combination of auxiliary learning with other measures for the sample efficiency problem, such as hierarchy or human feedback is an interesting frontier.

Appendix A

Agent Architecture





A. Overview of the architecture. The input image with 3 channels (RGB) goes through the shared convolutional encoder and is fed to each decoder: the core head for the policy and value function, the pose estimation head, the reward prediction head, and the local map prediction head.

B. Convolutional module. The module consists of three blocks, each outputing 16, 32, and 32 channels respectively. Each block is made up of a convolutional layer with stride 1 and 3x3 filters, followed by a max pooling layer with 3x3 filters and stride 2, followed by two residual blocks. Each residual block consists of a relu nonlinearity, a convolutional layer with 3x3 filters, a relu nonlinearity, and another convolutional layer with 3x3 filters. The output of each the residual block is summed with its input to form the input for the next residual block. The output of the residual module is nonlinearied with a relu activation and finally passes through a fully connected layer of 256 neurons with a relu nonlinearity.

C. Core decoder outputing the policy and value function. The module consists of an LSTM layer with 256 neurons. Its input is made up of the past action and reward, the output from the convolutional module, the output from the auxiliary LSTM, and the pose estimation output.

D. Pose estimation decoder. The output from the convolutional module is the input to the LSTM unit with 128 neurons. The output is splitted into two heads. One for the position decoder consisting of a multilayer perceptron with 128 and 3 neurons respectively and a relu activation after both layers. The reason the output layer has a relu activation is to force the predictions to the positive domain as the position can only be positive in the implemented environment. The final layer of the orientation head has 12 neurons and a softmax activation for outputing the belief about the binned orientation.

C. Reward prediction decoder. The module consists of a multilayer perceptron with a layer of 64 and a layer of 2 neurons with a relu and a softmax activation respectively. We hope to prevent the network from becoming overconfident in predicting zero rewards as the softmax nonlinearity does not allow zero outputs. The input comes from the same LSTM layer as the pose estimation decoder.

F. Local map prediction decoder. The module has two fully connected layers with 256 neurons with relu nonlinearities. The second layer takes in the pose estimation as additional input. The final layer represents the pixels of the local map excerpt which in our experiments is 20x20. The final layer is activated with a sigmoid activation function to output the belief about the structure of the map excerpt (0 for unobstructed and 1 for obstructed).

Appendix B

Additional Results

In this appendix, we present the individual results to the presented experiments. These results show four different seeds, the average to which has been presented in the main section of the thesis.

B-1 Adaptive Auxiliary Weights

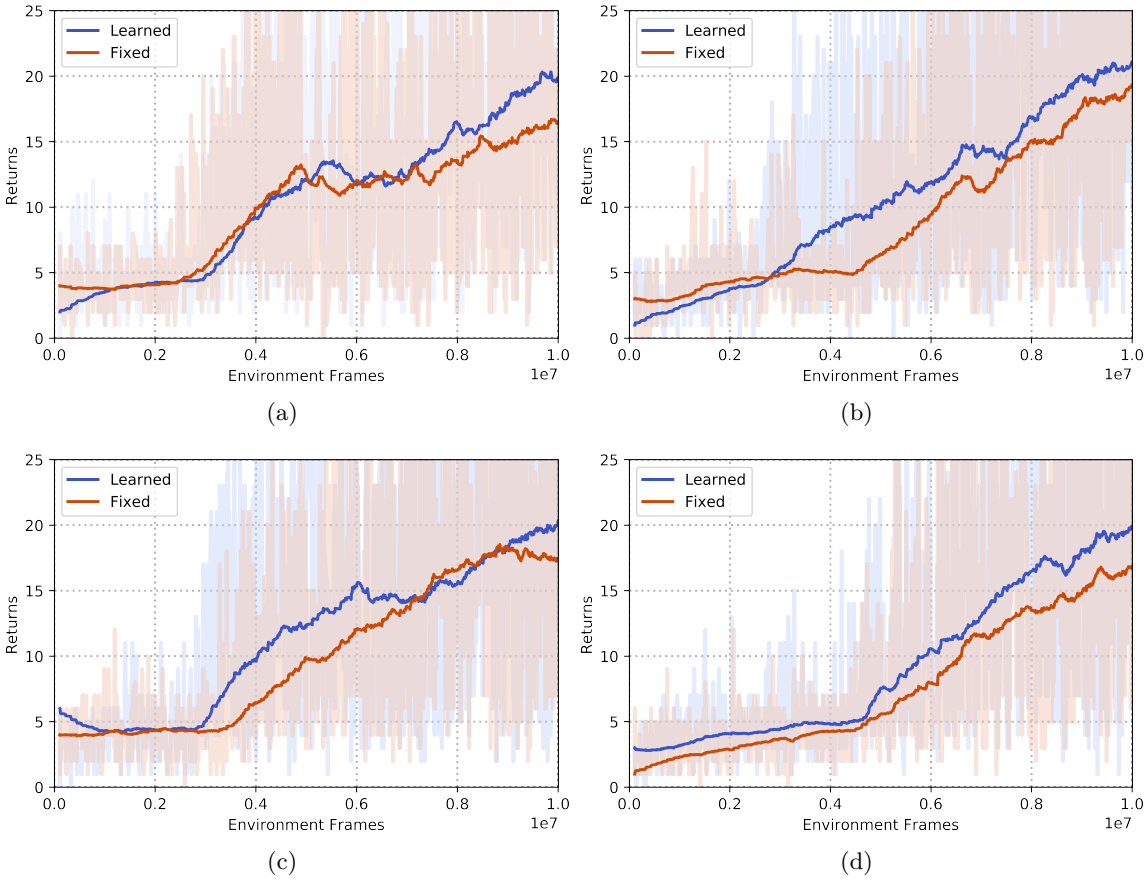


Figure B-1: Individual results adaptive auxiliary weights. Only the pose estimation and local map prediction losses are utilized. The fixed weight setting entails $\lambda_i = 1$.

B-2 Baseline Comparison

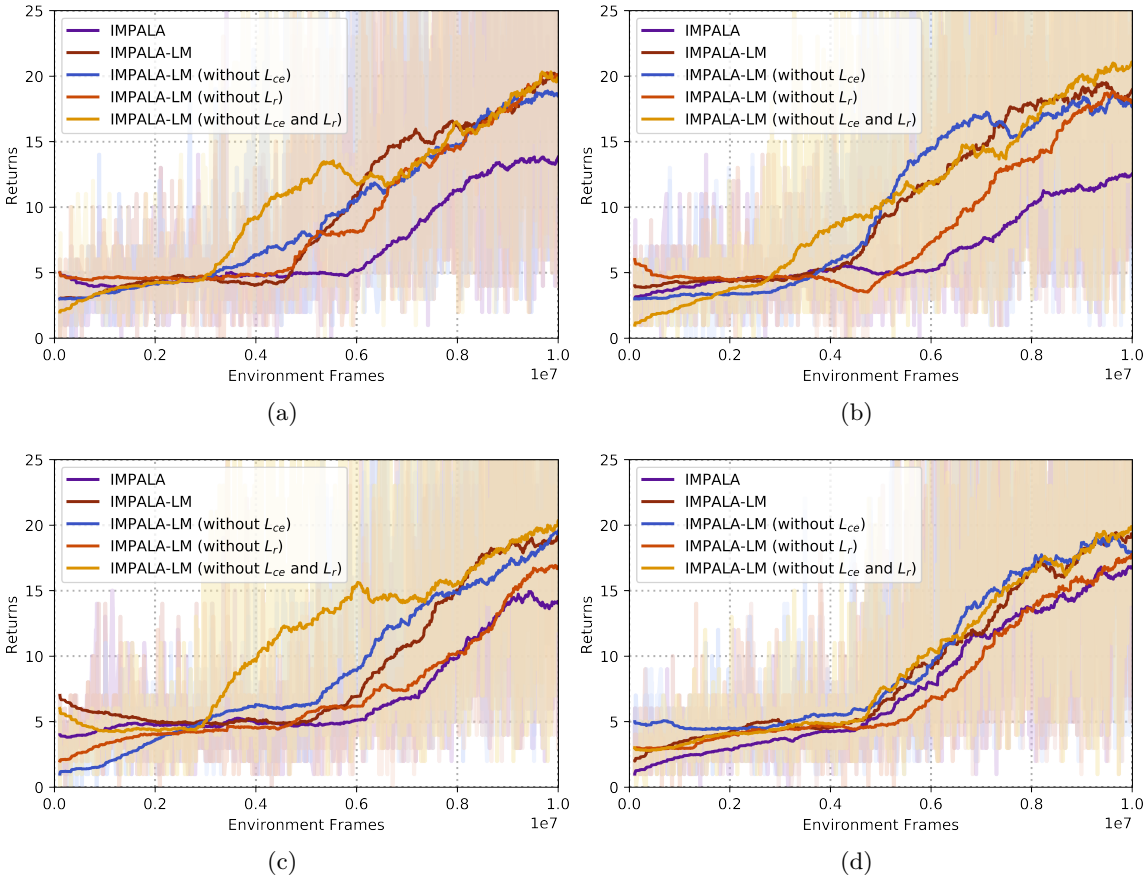


Figure B-2: Individual results baseline comparison. All IMPALA-LM implementations are with learned auxiliary weights.

B-3 Exploration Trajectory

In this section, we present trajectory examples to demonstrate the behavior of the agent in the environment.

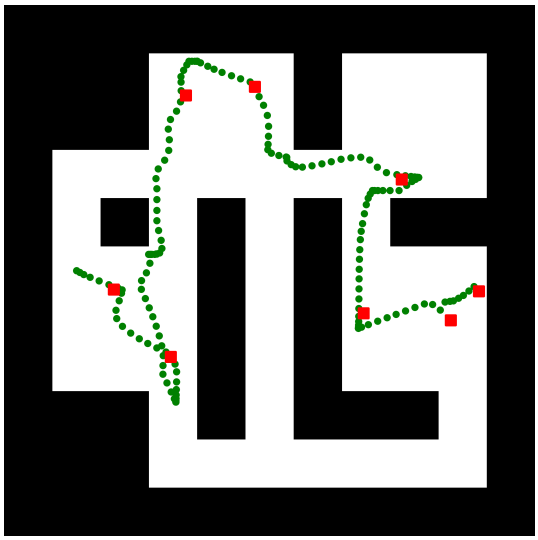


Figure B-3: Trajectory of IMPALA-LM in Test 1. In this example, the agent manages to find all 8 targets in the environment.

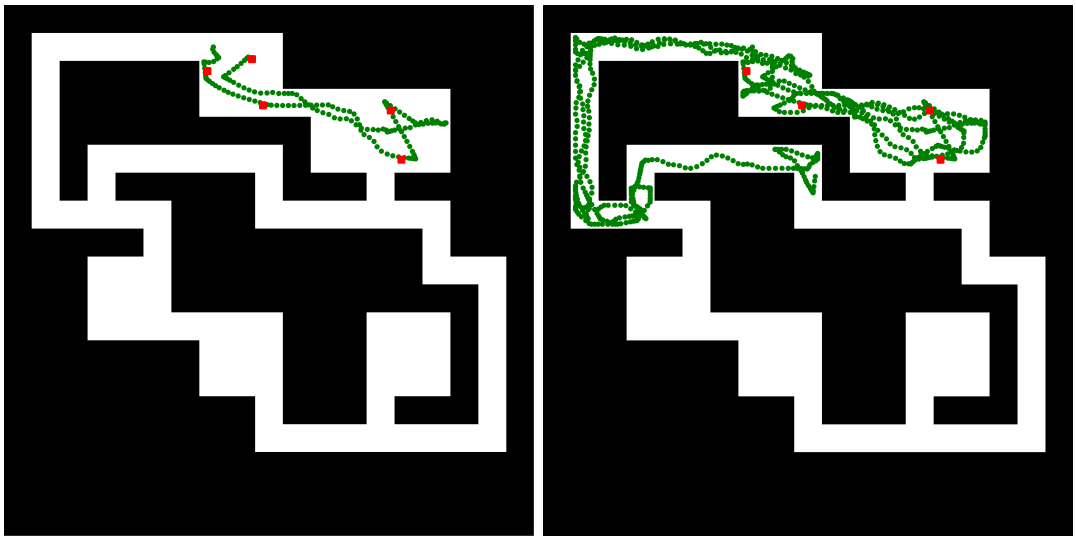


Figure B-4: Trajectory of IMPALA-LM in Test 2. This example demonstrates that while the agent manages to find 5 targets in the beginning (left), it struggles to get more feedback signals and starts to wander around until the end of the episode (right).

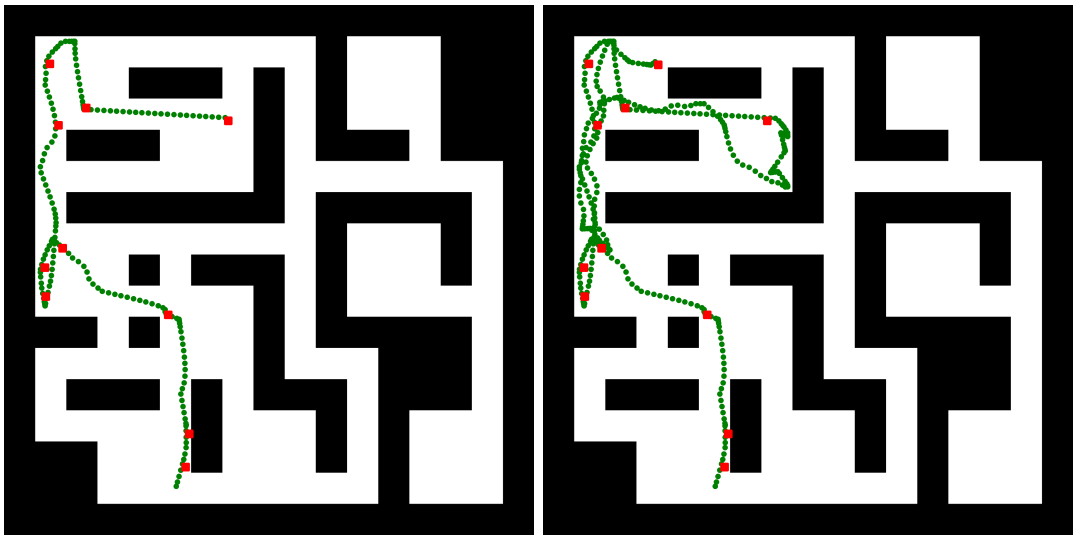


Figure B-5: Trajectory of IMPALA-LM in Test 3. This example shows that the agent recognizes targets and directly traverses towards them efficiently (left). As the agent tries to find the next target, it observes a target that is in its sight, and opt to move towards it. However this target is further away than a target that was not directly in its sight. As a result, the agent struggles to find the obstructed target located behind its back and makes unnecessary movements before being able to find the next target (transition from left to right).

References

[1] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151, IEEE, 1997.

[2] K. Verbiest, S. A. Berrabah, and E. Colon, "Autonomous frontier based exploration for mobile robots," in *International Conference on Intelligent Robotics and Applications*, pp. 3–13, Springer, 2015.

[3] A. Topiwala, P. Inani, and A. Kathpal, "Frontier based exploration for autonomous robot," *arXiv preprint arXiv:1806.03581*, 2018.

[4] C. Gomez, A. C. Hernandez, and R. Barber, "Topological frontier-based exploration and map-building using semantic information," *Sensors*, vol. 19, no. 20, p. 4595, 2019.

[5] M. N. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Engineering Practice*, vol. 15, no. 4, pp. 435–445, 2007.

[6] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[7] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular slam," in *European conference on computer vision*, pp. 834–849, Springer, 2014.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[11] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[12] R. A. Howard, "Dynamic programming and markov processes.," 1960.

[13] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[14] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[15] O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans, "Bridging the gap between value and policy based reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 2775–2785, 2017.

[16] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063, 2000.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, p. 533, 1986.

[20] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[21] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[22] J. Hinton, "Neural networks for machine learning course."

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[25] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[26] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[28] I. Popov, N. Heess, T. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," *arXiv preprint arXiv:1704.03073*, 2017.

[29] J. Clark and D. Amodei, "Faulty reward functions in the wild," 2016. URL: https://blog.openai.com/faulty-reward-functions/.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[31] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.

[32] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] S. C. Suddarth and Y. Kergosien, "Rule-injection hints as a means of improving network performance and learning time," in *European Association for Signal Processing Workshop*, pp. 120–129, Springer, 1990.

[35] R. Sutton, J. Modayil, M. Delp, T. Degris, P. Pilarski, A. White, and D. Precup, "Horde : A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction categories and subject descriptors," vol. 2, 01 2011.

[36] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[37] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[38] B. Kartal, P. Hernandez-Leal, and M. E. Taylor, "Terminal prediction as an auxiliary task for deep reinforcement learning," *CoRR*, vol. abs/1907.10827, 2019.

[39] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "Agent modeling as auxiliary task for deep reinforcement learning," *CoRR*, vol. abs/1907.09597, 2019.

[40] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, *et al.*, "IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[41] D. Precup, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, p. 80, 2000.

[42] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *CoRR*, vol. abs/1603.05027, 2016.

[44] G. Brunner, O. Richter, Y. Wang, and R. Wattenhofer, "Teaching a machine to read maps with deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[45] A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5974–5983, 2017.

[46] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "ViZDoom: A Doom-based AI research platform for visual reinforcement learning," in *IEEE Conference on Computational Intelligence and Games*, (Santorini, Greece), pp. 341–348, IEEE, Sep 2016. The best paper award.

[47] D. Pérez-Liébana, K. Hofmann, S. P. Mohanty, N. Kuno, A. Kramer, S. Devlin, R. D. Gaina, and D. Ionita, "The multi-agent reinforcement learning in malmö (marlö) competition," *CoRR*, vol. abs/1901.08129, 2019.

[48] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[49] C. Beattie, J. Z. Leibo, D. Teplyashin, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen, "Deepmind lab," *CoRR*, vol. abs/1612.03801, 2016.

[50] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, *et al.*, "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning," *arXiv preprint arXiv:1807.01281*, 2018.

[51] X. Lin, H. Baweja, G. Kantor, and D. Held, "Adaptive auxiliary task weighting for reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 4773–4784, 2019.

[52] C. Atkinson, B. McCane, L. Szymanski, and A. Robins, "Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting," *arXiv preprint arXiv:1812.02464*, 2018.

[53] Y. Du, W. M. Czarnecki, S. M. Jayakumar, R. Pascanu, and B. Lakshminarayanan, "Adapting auxiliary losses using gradient similarity," *arXiv preprint arXiv:1812.02224*, 2018.