Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Datateknik
2022 | LIU-IDA/LITH-EX-A–2022/001–SE

Learning to Search for Targets

– A Deep Reinforcement Learning Approach for Visual Search in Unknown Environments

Inlärd sökning efter mål

Oskar Lundin

Supervisor : Sourabh Balgi
Examiner : Jose M. Peña

External supervisor : Fredrik Bissmarck

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

To do...

Acknowledgments

To do...

Contents

List of Figures

List of Tables

1 Introduction

In this thesis project, the problem of searching for target objects in unknown but familiar environments is addressed. This chapter presents the motivation behind the project, the research questions that are addressed, and the delimitations.

1.1 Motivation

The ability to visually search for things in an environment is fundamental to intelligent behavior. We humans are constantly looking for things, be it be it the right book in the bookshelf, a certain keyword in an article or blueberries in the forest. In many cases, it is important that this search is strategic, efficient, and fast. Animals need to quickly identify predators, and drivers need to be able to search for pedestrians crossing the road they are driving on.

Automating visual search is of great interest for several reasons. Visual search is crucial for applications such as search and rescue, surveillance, fire detection, etc. Autonomous vehicles can both reduce risk and potentially exhibit more intelligent searching behavior than human-controlled ones.

However, while visual search is often seemingly effortless to us humans, it is a complex process. Attempts to understand and recreate human visual search in machines has been a big challenge [25]. At the root of the visual search problem is partial observability. A searcher can only perceive, or pay attention to, a limited region of the searched environment at once. Therefore which regions to observe and in what order becomes an important decision.

How humans and animals search for things has been studied extensively in neuroscience [25, 64, 48]. When we search, we use features of the environment to guide our attention [65, 25]. For example, we know to look for berries at the forest floor, and not to look for boats on land. Furthermore, search is not purely reactive but involves the use of memory. We also use memory to take the history of the search into account when deciding where to move our attention [65].

Such features can in some cases be quite subtle and difficult to pick up, even for humans. Manually engineering guidance in accordance with these features can be expensive, especially if a searching system should be deployed in many different environments. If one could instead learn a good searching strategy from a limited set of sample environments this would be circumvented. Such a system could be taught to search in arbitrary environments without the use of manually encoded environment-specific rules.

Reinforcement learning [60] (RL) is a paradigm that is suited for learning mappings from sensor values to actions. In recent years, RL has been combined with deep learning [29] with tremendous success. It has been used to master arcade games [47], board games [58], and even complex real-time strategy games [63]. Several works have also applied RL to tasks involving embodied agents with visual input [42, 45, 71, 43]. This makes it interesting to see if RL can also be applied to visual search.

1.2 Aim

The aim of this thesis is to investigate how an intelligent agent that learns to search for targets using visual input can be implemented with deep reinforcement learning. Such an agent should learn the characteristics of the environments it is trained on and utilize this knowledge to search strategically in unseen environments. Specifically, we consider scenarios where the agent can only observe a small portion of its environment at any given time through an camera whose movements are unrestricted. The agent has to actively choose where to look in order to gain new information about the environment.

We postulate that an effective searcher learns how the distribution of targets and prioritizes regions where they are more likely according to previous experience. The distribution of targets may be correlated with the appearance of the searched scene. A good searcher should integrate information over time to build an internal representation of the environment and use it to make informed decisions. The agent should be able to search the environment exhaustively while avoiding visiting the same region twice. Finally, the agent should be able to locate multiple targets while minimizing the number of steps taken.

If such a system is to be trained and deployed for a real-world task, there is likely a limited set of samples to learn from. Therefore it is also of interest to investigate how many samples are required to infer how to effectively search in similar environments. While similar problems have been addressed in the past, both with learning agents [42, 43, 28, 16] and non-learning agents [57, 26], our impression is that this is the first work to address learning to search in unseen environments where emphasis is placed on how arbitrary visual cues can guide search. Our contributions are as follows:

- We provide a set of environments to train and evaluate visual search agents.
- We propose two approaches for solving the visual search task with reinforcement learning.
- We evaluate the performance of the proposed methods to a set of common baseline agents.
- We investigate how well each agent is able to generalize to unseen environments.
- We discuss our method in terms of applicability and usefulness.

1.3 Research Questions

This thesis will address the following questions:

1. How can an agent that learns to intelligently search for targets be implemented with reinforcement learning?
2. How does the learning agent compare to random walk, greedy search, and a human searcher with prior knowledge of the searched environment?
3. How well does the learning agent generalize from a limited number of training samples to unseen in-distribution scene samples?

1.4 Delimitations

We focus on the behavioral and decision-making aspects of the presented problem, and delimit ourselves from difficult detection problems. For this reason, targets will deliberately be made easy to detect once visible. Furthermore, we make the assumption that the searched environment is static. The appearance of the environment and the location of the targets does not change from one observation to the next. Finally, we are specifically interested in deep reinforcement learning approaches.

2 Theory

This chapter introduces background and related work.

2.1 Background

In this section, we give background to the tackled problem and outline the theory behind our approach. Sections 2.1.1 and 2.1.2 provide perspective to the problem from neuroscience and machine perception respectively. Section 2.1.3 overviews the theory behind function approximation with neural networks, and some common neural network architectures. Section 2.1.4 summarizes the foundations of reinforcement learning and deep reinforcement learning.

2.1.1 Visual Search and Attention

The perceptual task of searching for something in a visual environment is usually referred to as *visual search* [64]. The object or feature that is being searched for is referred to as the *target*, and the other objects or features in the environment as *distractors*. This task has been studied extensively in psychology and neuroscience.

Two big limitations of performance in visual search are processing power, and limited observability. Processing high-dimensional sensory input like images is expensive, and environments are often too large and complex to view all at once. An observer that searches a scene for targets has to direct its *visual attention* to one region at a time. Humans scan environments by directing their gaze (*overt attention*) and shifts of attention in the current visible region (*covert attention*) [36]. While covert attention tends to be reactionary, overt attention usually integrates more features over time. In this work we focus specifically on the former, moving the gaze to bring targets into view.

Humans control overt attention through both eye movements and head movements. Eye movement is almost instant between locations, and the cost of directing attention with eye movements is constant regardless of distance. This is not true for head movements, and in general not true for robotic systems: movements induced by motors tend to come at a cost that is proportional to the distance moved, both in time and energy. The cost of directing overt attention means that there is a need to do so strategically.

If the searched environment is a random field, visual search is akin to a random process [48]. Experiments in humans have shown that search in featureless environments is consistent with

random walk with no memory. Optimal search algorithms in such environments would be exhaustive, such as those found in coverage path planning [27]. Humans have also been shown to use the history of observations to improve visual search efficiency. Simple memory mechanisms, like some inhibition-of return mechanism [36] that prevents searching visited locations twice, improve search time in random fields considerably.

Natural environments are in fact seldom completely random, but instead tend to exhibit some structure. When finding targets, there is usually something about this structure that can be utilized to improve search performance. Such regularities can learned from past experience and used during future searches. Improvements in human search performance have been measured when certain locations have higher probabilities of containing targets [25, 65]. Furthermore, if targets usually co-occur with other visible elements they tend to be easier to find [25, 65].

2.1.2 Active Vision and Object Search

Much of past and present research in machine perception involves a passive observer which samples and perceives images from a fixed distribution. Animal perception, in contrast, is active – we do not only see, but also decide where to look. In the *active vision* paradigm, an the observer has some control of its sensory input. [3]

Active vision, and *active perception* in general, is a problem of intelligent data acquisition. An active observer must control its sensory inputs to constrain the interpretation of its environment. One of the difficulties of active perception problems is that they are scene and context dependent. A thorough understanding of the data acquisition parameters and the goal of the visual processing is needed. [10]

In active vision, searching for objects with a camera in unknown environments is referred to as *object search* or *active object localization*. A searching observer has to both recognize and localize its targets, while controlling its sensory input. To search effectively, it must also model its environment and perform path planning. [18]

Solving the object search problem optimally involves determining a sequence of camera controlling actions that maximizes the probability of finding the target while satisfying a cost constraint. The state of the searcher is uniquely determined by the control parameters of the camera. Actions that adjust these parameters and adjust the observed region come at some cost in time or energy. Finally, the agent has some prior knowledge of the probability of targets which it updates after each observation. Finding an optimal solution in three dimensional search spaces has been shown to be NP-complete, necessitating approximate solutions. [68, 5]

2.1.3 Deep Learning

Deep learning is a family of techniques in which hypothesis are represented as computation graphs with tunable weights. The computation graphs are inspired by biological neurons in the brain and are referred to as *neural networks*. Deep neural networks consist of *nodes* arranged in *layers*: one input layer, zero or more hidden layers and one output layer. Each layer receives an input *representation* [12] from the previous layer and outputs a transformed representation to the next layer. Given some input, a neural network optimizes its output representation with regard to some criterion. Usually, a loss function $\mathcal{L}$ is minimized by updating the weights $\mathbf{w}$ of the network with some variant of *gradient descent* with learning rate α:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{2.1}$$

The only requirement on the functions computed by each node is that it is differentiable. As long as this holds, layers can be stacked arbitrarily and the gradients can be computed with the chain rule. This way, errors in the output can be passed back through the network (*back-propagation*) and used to update the weights. [55, 29]

The architecture of a neural network imposes some bias onto the learning that its expected to be useful for generalizing to unseen samples. We now describe three neural network architectures that will be used in this work.

Feedforward Neural Network

A feed-forward neural network, also known as a multi-layer perceptron (MLP) [29], only has connections in one direction. Each node in the network receives inputs from its predecessors and outputs the result of a function of those inputs. The output y of each node is usually computed by taking the weighted sum of its inputs x and applying some non-linear function

$$y_j = g_j(\mathbf{w}_j^T \mathbf{x}), \tag{2.2}$$

where y_j is the output of node j, g_j is a non-linear *activation function*, $\mathbf{w}_j$ is the vector of weights leading into node j, and $\mathbf{x}$ is the vector of inputs to the node. By convention, each layer also has some *bias* that allows the total weighted input to g_j to be non-zero even when the outputs from the previous layer are zero. The bias is included as an extra input x_0 fixed to 1, and an extra tunable weight $w_{0,j}$. The non-linearity ensures that a network with at least two layers can approximate any continuous function. [55]

Convolutional Neural Network

Convolutional neural networks (CNNs) contain spatially local connections. They have patterns of weights, called *kernels*, that are replicated across units in each layer. With some input vector $\mathbf{x}$ of size n and a vector kernel $\mathbf{k}$ of size l, the (discrete) convolution operation $\mathbf{z} = \mathbf{x} * \mathbf{k}$ is defined as

$$z_i = \sum_{j=1}^{l} k_j x_{j+1-\frac{l+1}{s}}, \tag{2.3}$$

where s is the *stride*. This operations can be generalized up to more than one dimension, such as 2 dimensions for images and 3 dimensions for volumes. With multiple input channels, kernels are stacked into a *filter*. The outputs of each kernel are then summed over, giving one output channel per filter.

There are several advantages to using CNNs for structured input data where neighboring values are correlated. Kernels are smaller than the input, which means that fewer parameters have to be stored. These *sparse interactions* give CNNs reduced memory requirements, as well as improved statistical and computational efficiency.

Furthermore, the same parameters are also used for more than one function in the CNN. *Parameter sharing* across input locations mean that layers in a CNN have *equivariance* to translation. The output of one kernel is the same regardless of the input location. This property of CNNs is useful for images where similar features may be useful regardless of their location in the input. [29]

Recurrent Neural Network

Recurrent neural networks (RNNs) extend feed-forward networks by allowing cycles in the computation graph. Each cycle has a delay so that some *hidden state* from the previous computation is used as input to the current computation. A recurrent layer with input $\mathbf{x}_t$, output $\mathbf{y}_t$ and hidden state $\mathbf{z}_t$ is defined by

$$\begin{aligned} \mathbf{z}_t &= f_{\mathbf{w}}(\mathbf{z}_{t-1}, \mathbf{x}_t) \\ \mathbf{y}_t &= g_y(\mathbf{W}_{z,y}, \mathbf{z}_t), \end{aligned} \tag{2.4}$$

where $f_{\mathbf{W}}$ is the update process for the hidden state and g_y is the activation function for the hidden layer. This model can be turned into a feed-forward network over a sequence of input vectors $\mathbf{x}_1, \ldots \mathbf{x}_T$ and observed outputs $\mathbf{y}_1, \ldots, \mathbf{y}_T$ by *unrolling* it for T steps. The weights are shared across all time steps. This means that RNNs can operate on inputs of arbitrary lengths. The hidden state is used as a summary of all previous items in the sequence. Thus, RNNs make a Markov assumption. [55]

In practice, conventional RNNs struggle with learning long-term dependencies. During back-propagation, gradients can tend to zero for long sequences, something known as the vanishing gradient problem [29]. An architecture that addresses this issue is long short-term memory (LSTM) [35]. LSTMs include a *memory cell* c in the hidden state that is copied from time step to time step, and three soft *gating units* that govern the information flow in the hidden state update process f. This makes LSTMs particularly useful for learning over long sequences.

2.1.4 Reinforcement Learning

Reinforcement learning (RL) [60] is a subfield of machine learning concerned with learning from interaction how to achieve a goal. This section introduces the fundamental concepts of RL.

Partially Observable Markov Decision Processes

The problem of learning from interaction to achieve some goal is often framed as a Markov decision process (MDP). A learning *agent* interacts continually with its *environment*. The agent takes the *state* of the environment as input, and select an *action* to take. This action updates the state of the environment and gives the agent a scalar *reward*. It is assumed that the next state and reward depend only on the previous state and the action taken. This is referred to as the *Markov* property. [37]

In an MDP, the agent can perceive the state of the environment with full certainty. For many problems, including the one we consider here, this is not the case. The agent can only perceive a partial representation of the environment's state. Such a process is referred to as a partially observable Markov decision process (POMDP). A POMDP is formally defined as a 7-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma \rangle$, where

- $\mathcal{S}$ is a finite set of states,
- $\mathcal{A}$ is a finite set of actions,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Pi(\mathcal{S})$ is a state-transition function,
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function,
- Ω is a finite set of observations,
- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ is an observation function, and
- $\gamma \in [0, 1]$ is a discount factor.

Assume that the environment is in state $s_t \in \mathcal{S}$, and the agent selects action $a_t \in \mathcal{A}$. Then, $T(s_t, a_t, s_{t+1})$ is the probability of ending in state s_{t+1} and $r_t = R(s_t, a_t)$ is the expected reward gained by the agent. The agent also receives an observation $o_t \in \Omega$ with probability $\mathcal{O}(s_{t+1}, a_t, o_t)$. [37] Figure 2.1.4 illustrates the interaction between agent and environment.

The agent and environment interact over a sequence of discrete time steps $t = 0, 1, \ldots, T$, giving rise to an *episode* of length T. At each time step t, the goal of the agent is to select the action that maximizes the expected *discounted return*:

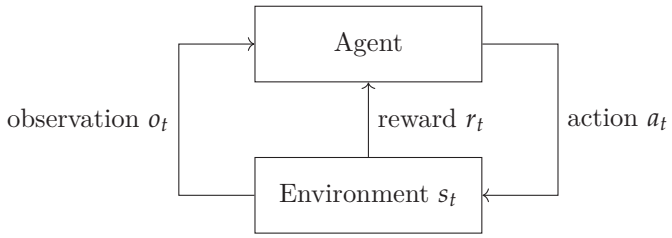

Figure 2.1: Interaction between agent and environment in a partially observable Markov decision process.

$$\mathbb{E}\left[\sum_{k=0}^{T} \gamma^{k-t-1} r_k\right] \tag{2.5}$$

Since the agent receives partial observations of the environment's state, it has to act under uncertainty. Planning in a POMDP is undecidable, and solving one is often computationally intractable. Approximate solutions are more common, where the agent usually maintains an internal *belief state* [37] which is acts on. The belief state summarizes the agent's previous experience and is therefore dependent on the full *history* of actions and observations. It does not need to summarize the whole history, but generally only the information that helps the agent maximize the expected reward. From here on we will use the belief state and the environment state s interchangeably.

Policies and Value Functions

The behavior of the agent is described by its *policy*. A policy π is a mapping from perceived environment states to actions. Policies are often stochastic and specify probabilities for each action, with $\pi(a|s)$ denoting the probability of taking action a in state s. [60]

Most RL solutions methods also approximate a *value function*. A value function V_π estimates how good it is to be in a state. The value function $V_\pi(s)$ is the expected (discounted) return when starting at state s and following policy π until the end of the episode. There are two common alternative value functions: The *quality function* $Q_\pi(s,a)$ gives the value of state s under policy π where a is the first action taken. Given a quality function, *action-value* methods choose the action greedily at every state as $arg\,max Q_\pi(s,a)$. The *advantage function* $A_\pi(s,a)$ instead represents the relative advantage of actions, $A_\pi = Q_\pi - V_\pi$. [60]

For problems with large state and action spaces, it is common to represent value functions with *function approximation*. In such cases, it is common to encounter states that have never been encountered before. This makes it important that the estimated value function can generalize from seen to unseen states. With examples from the true value function, an approximation can be made with supervised learning methods. We write $\hat{V}(s,\mathbf{w}) \approx V_\pi(s)$ for the approximate value of state s with some weight vector $\mathbf{w} \in \mathbb{R}^d$. [60]

An alternative to action-value methods is to approximate the policy itself. *Policy gradient* methods learn a parametrized policy that select actions without a value function. We denote a parametrized policy as $\pi(a|s,\boldsymbol{\theta})$ with $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ as the parameters to the policy. The policy parameters are usually learned based on the gradient of some performance measure $L(\boldsymbol{\theta})$. As long as $\pi(a|s,\boldsymbol{\theta})$ is differentiable with respect to its parameters, the parameters can be updated so as to optimize for the objective. [59]

Advantages of policy parametrization over action-value methods include stronger convergence guarantees [59] and more flexibility in parametrization [60]. In practice, value functions are often still used to learn the policy parameter, but they are not needed for action selection. Such methods are called *actor-critic* methods, with actor referring to the learned policy and

critic referring to the learned value function. In these cases, there might also be some overlap between the weights $\mathbf{w}$ of the value function estimate and $\boldsymbol{\theta}$ of the policy estimate.

One important aspect of function approximation is its interplay with partial observability. If there is a state variable that is not observable, as for partially observable environments, then the parametrization can be chosen such that the approximate value does not depend on that state variable. Because of this, function approximation is applicable to the partially observable case. [60]

Design Challenges

One of the challenges that arises in reinforcement learning is the exploration-exploitation trade-off. An RL agent should *exploit* knowledge gained from previous experiences and prefer actions that has yielded reward in the past. It should also *explore* in order to learn better actions to take in the future. Agents that fail to both exploit and explore will lead to failure at the task, and striking a good balance between the two is non-trivial. [60]

Another challenge is the design of reward signals. For some tasks, like certain video games, the objective is simply to maximize the score obtained. In this case there is an inherent reward signal and the agent achieves its task simply by maximizing this inherent signal. Other times, we have a task we want the agent to solve and have to design a reward signal around that task. Designing rewards is not straight-forward and can often have unintended effects [60]. Special care has to be taken to ensure that the reward encourages the desired behavior.

Here, the problem of *sparse rewards* also comes into play. The agent has to be reward frequently enough to allow it to achieve its goal once. Often it has to incentivize it to achieve its goal efficiently, with multiple different starting conditions. If rewards are too sparse, the agent may explore aimlessly and take too long to find achieve its goal. If the received reward is temporally distant from the action that caused it, the agent may have difficulty connecting the two. This is known as the *credit assignment problem* [41].

In practice, rewards are often designed through trial-and-error. Through *reward shaping* [40], the reward is designed so as to guide the agent towards achieving its goal by giving additional rewards along the way. Several iterations of a reward signal are tried until one yields expected and sufficient results.

Deep Reinforcement Learning

As mentioned in Section 2.1.4, policies and value functions are often approximated. Neural networks have good properties for function approximation and have been used for RL with success. One early example is TD-Gammon [61], a neural network trained with RL that reached expert Backgammon performance in 1995.

More recently, the successes of deep learning have bled over into the field of RL. In 2015, Mnih et al. [47] extend [46] and introduce DQN, which combines deep neural networks with RL. DQN successfully plays Atari games using only visual input. It approximates the quality function $q(s,a)$ with a CNN architecture, and selects actions greedily. To incorporate some memory, images from the 4 previous time steps are stacked and used as input to the neural network. The input is fed through three convolutional layers and a hidden fully connected layer, all with ReLU activation functions. The output layer has one output for each valid action, representing Atari controller buttons.

DQN inspired many several follow-up works which use deep neural networks to approximate policy and/or value functions. Such methods are often referred to as deep reinforcement learning (deep RL) methods.

Proximal Policy Optimization

Proximal policy optimization (PPO) [56] is a family of policy-gradient algorithms that have turned out to strike a good balance between simplicity, stability and ease of tuning while achieving strong results on several tasks [56, 32, 20, 63, 6].

PPO alternates between sampling agent-environment interactions over T time steps and optimizing the policy using those interactions. The parameters $\boldsymbol{\theta}$ of the policy π are updated with the objective

$$\mathcal{L}_{\text{CLIP}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t \right]. \tag{2.6}$$

The expectation $\hat{\mathbb{E}}$ indicates the empirical average over a finite batch of samples collected under the old policy with parameters θ_{old}. Here, $r_t(\theta) = \frac{\pi(a_t|s_t,\boldsymbol{\theta})}{\pi(a_t|s_t,\boldsymbol{\theta}_{\text{old}})}$ and $\hat{A}_t$ is an estimate of the advantage function. The clip range ϵ is used to clip the surrogate objective, putting a pessimistic bound on the product of the probability ratio and the advantage estimate. This in turn ensures that the size of the policy updates is limited.

In an actor-critic approach, the advantage is estimated over using a learned state value function $V(s)$ as

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{2.7}$$

where $\delta_t = r_t + \gamma V(s_t+1) - V(s_t)$. If the policy and value functions estimations share parameters, for example in a multi-headed neural network architecture, a loss function that combines the policy loss and the value function error term must be used. It is also common to introduce an entropy bonus to ensure sufficient exploration. The full PPO objective for an actor-critic approach is

$$\mathcal{L}_t(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\mathcal{L}_{\text{CLIP}}(\boldsymbol{\theta}) - c_1\mathcal{L}_{\text{VF}}(\boldsymbol{\theta}) + c_2 S[\pi](s_t, \boldsymbol{\theta}) \right] \tag{2.8}$$

where the coefficients c_1 and c_2 are hyperparameters determining the impact of the value loss and entropy bonus, S is an entropy bonus and $\mathcal{L}_{\text{VF}}$ is a squared-error loss in the value output. Algorithm 2.1.4 shows the steps of the PPO algorithm.

Algorithm 1 Proximal Policy Optimization

```
for iteration = 1, 2, ... do
    for actor = 1, 2, ..., N do
        run policy π_θold for T time steps
        compute advantage estimates Â_1, ... Â_T
    end for
    optimize L wrt θ, with K epochs and mini-batch size M ⩽ NT
    θ_old ← θ
end for
```

2.2 Related Work

Works that consider tasks that are related to searching for targets in unknown environments can be found in several fields under many names. In this section we survey some of the more relevant ones, focusing on those that employ RL methods.

2.2.1 Search with Reinforcement Learning

There are several rigorous attempts to implement robotic object search systems using non-learning methods. Forssén et al. [26] implement a mobile robot that searches for specific objects within a cluttered environment. Their robot explores by moving towards unexplored

regions and looks around to identify potential objects. Each objects is examined from several perspectives and ranked by probability of being the queried object. Shubina and Tsotsos [57] model the search problem as in [68], and solve with a greedy action selection strategy. They do not take environment appearance into account, and rely on prior knowledge of target distribution being provided as a spatial probability map. Some works also take appearance and semantics of the environment into account to search more efficiently [7, 8].

These systems solve more complex tasks than the one we consider, such as handling multiple target object categories and navigating in obstructed environments. However, they are also restricted by their reliance on human knowledge to act well in their environments. While reinforcement learning is not among the more traditional solution methods for active vision tasks [18], learning systems that are less dependent on expert knowledge have the potential to be more generally applicable.

Sequential Visual Attention

Minut and Mahadevan[42] propose an sequential model of selective visual attention for visual search tasks. An agent is tasked with finding a particular object in a scene A policy for controlling a fixed pan-tilt-zoom camera is learned using reinforcement learning. The goal of the agent is to aim the camera at the region where a target is most likely to be found. It has to decide where to fixate next based on visual information only. Despite being limited to a single environment, this is an early example of visual search modelled as an RL problem.

Mnih et al. [45] take inspiration from visual attention and foveated vision found in humans and propose to use a similar mechanism for computer vision tasks. Applying a CNN to a large images can be expensive, as the complexity scales linearly with the number of images pixels. They propose a recurrent model that extracts information from images by adaptively selecting a sequence of smaller regions to process at high resolution. At each time step, the agent receives an image observation of the environment. Through actions, the agent selects a limited region of this observation to view in high resolution. It is given a reward that is dependent on the task the agent should perform. It can access this image via a bandwidth-limited sensor which it focuses on a limited region. The agent maintains an internal state using an LSTM layer. Though the model is not differentiable, it is trained using RL with a policy gradient method. The authors evaluate the agent for image classification and a game-like task. They find that the model outperforms a similar convolutional architecture for cluttered object classification tasks. This is attributed to its ability to focus its attention on important regions.

Active Object Detection

In computer vision, *object detection* is the task of detecting semantic objects of a certain class in images. Detecting an object entails *recognizing* that it is present in an image, and *localizing* it by determining its bounding box. State of the art object detection use deep learning techniques and usually involve a deep convolutional architecture to recognize objects [70]. For localization, a region proposal process considers the whole image and returns a set of bounding box candidates. The object recognizer is then run on each region proposal. Region proposals are akin to visual attention, as they limit the region of the image that is processed by the recognizer.

Although we do not focus on difficult recognition problems in this work, object localization is highly relevant. The difference is that images in object detection are passively sampled - they are drawn from some distribution and all are independent. Furthermore, the whole scene that is searched for objects is visible in the image. In *active object detection*, the analyzed images are instead chosen so as to help the detector perform its task better.

Caicedo and Lazebnik [16] propose to use deep reinforcement learning for active object localization in images where the object to be localized is fully visible. An agent is trained to successively improve a bounding box using translating and scaling transformations. They use

a reward signal that is proportional to how well the current box covers the target object. An action that improves the region is rewarded with +1, and given a punishment of -1 otherwise. They find that this reward communicates more clearly which transformations keep the object inside the box and which take the box away from the target. When there is no action that improves the bounding box, the agent may select an indication action (which would be the only action that does not give a negative reward) which resets the box. This way the agent may select additional bounding boxes. Each indication action modifies the environment by marking it so that the agent may learn to not select the same region twice.

A similar work by Ghesu et al. [28] present an agent for anatomical landmark detection trained with deep RL. Different from [16] is that the entire scene is not visible at once. The agent sees a limited region of interest in an image, with its center representing the current position of the agent. The actions available to the agent translate the view up, down, left and right. A reward is given to the agent that is equal to the supervised relative distance-change to the landmark after each action. Three datasets of 891 anatomical images are used. The agent starts at random positions in the image close to the target landmark and is tasked with moving to the target location. While achieving strong results (90% success rate), the scenes and targets are all drawn from a distribution with low variance. Most real-world search tasks exhibit larger variance than a population of anatomical images of the human body.

Chen and Gupta [19] use a spatial memory for context reasoning in object detection. They argue that object detection systems require memory to perform well. Furthermore, they pose that this memory should capture the spatial layout of the scene in order to model object-object relationships. Their proposed memory remembers features and locations of past detections in an image and uses this to improve future ones. A CNN is used to extract features from the memory and used together with the original image as input to a standard region proposal network. The approach gives a small improvement over the standalone region proposal network.

Visual Navigation

Visual navigation is the process of determining a suitable path between a start point and destination point using visual input [13]. Visual navigation has been studied for use in both autonomous ground vehicles, where obstacles have to be avoided, as well as for unmanned aerial vehicles which typically don't have the problem of avoiding obstacles. The visual search problem can be considered analogous to a visual navigation in unrestricted environments. In recent years, several works have used deep RL to solve various visual navigation tasks.

Zhu et al. [71] create a model for target-driven visual navigation in indoor scenes with deep RL. An observer is given a partial image of its scene as well as an image of the target object, and is tasked with navigating to the object in the scene with a minimal number of steps. The agent moves forwards, backwards, and turns left and right at constant step lengths. They use a reward signal with a small time penalty to incentivize task completion in few steps. They compare their approach to random walk and the shortest path and achieve promising results. This setup is quite similar to the one considered in this report, but the authors make a few assumptions that we do not. They a set of 32 scenes, each of which contain a fixed number of object instances. They focus on learning spatial relationships between objects in these specific scenes, and have scene-specific layers to achieve this. Thus, while they show that they can adapt a trained network to a new scene, their approach is unable to zero-shot generalize to new scenes.

Ye et al. [67] propose an alternative architecture to [71] which, in addition to an image of the camera view and the target object, uses the output of an object recognition module to select actions. Specifically, the output of the localization component is fed as input. This allows the agent to know if the target object is in view, and where. The object recognition module is trained separately from the policy. While requires a set of images of potential target objects labelled with their class and location, it also illustrates how object recognition can

be offloaded to a separate module in reinforcement learning agents that benefit from such functionality.

Yang et al. [66] propose to use semantic scene priors to improve navigation towards objects in unseen scenes. The agent observes its environment through a camera. Prior knowledge of spatial relations between objects types is encoded as a graph, which is encoded with a graph convolutional network. Their agent is also told what object to look for in the form of a word embedding. The encoded object graph and word embedding is used as input together with the camera image to an actor critic network that selects actions that navigate towards the target object. As the agent collects experiences, it also updates its prior knowledge of spatial relationships between objects. It is shown that prior knowledge encoded in such a way improves the agents ability to navigate to objects.

Mnih et al. [44] use a recurrent policy with only RGB images to navigate in a labyrinth. 3D labyrinths are randomly generated, and an agent is tasked with finding objects in them. The same architecture as in [47] is used, but with 256 LSTM cells after the final hidden layer. At each episode, a maze is randomly generated with objects that give rewards then traversed scattered around. They train the agent using an actor-critic approach and manage to get the agent to successfully navigate in unseen but similar environments.

Mirowski et al. [43] propose a new approach to tackle a similar maze navigation problem as [44]. They introduce a new architecture, that in addition to an image, also observes the reward, velocity and the action from the previous time step. Observing the reward may help the agent learn features of desirable frames, but also limits the set of possible reward signals as the reward has to be present during testing. As in [44], the agent has a recurrent LSTM layer. The architecture is trained using an auxiliary depth prediction objective. The authors argue that understanding the depth of the searched environment helps the agent navigate it better. This agent is evaluated against three baselines on both static and randomly generated maze environments. It is found that an agent that only receives image observations performs worse than one that also has an LSTM layer. Adding the previous reward, action and velocity to the observations bring modest performance improvements. The depth prediction is found to improve performance in all environments, indicating that the correct auxiliary task can help learning.

Dhiman et al. [24] critically investigate deep RL for navigation, using a CNN and RNN architecture as in [31] and [43]. They ask whether deep RL algorithms are inherently able to gather and exploit environmental information for during navigation. Experimentally, they find that an agent is able to exploit environment information when trained and tested on the same map. When trained and tested on different maps, the agent does not seem to be able to exploit environment information to navigate more effectively. The authors use relatively small training set of less than 1000 samples, but do not consider that their agents could have overfit this this set. Finally, they find that the agent is not able to consistently find optimal navigation paths in seen environments when its starting position is randomized.

2.2.2 Memory Architectures

A recurring theme in the related work above is the use of memory architectures. In most cases where active sensing is involved, memory is a requirement for good performance. As mentioned in Section 2.1.4, effective behavior in partially observable processes usually requires some form of memory. Agents need to summarize the history of interactions with their environment for good action selection. Works that don't use memory rely on either full observability, like [16], or low variance so that search can be done reactively, like [28]. Memory is intuitively important for the search task we consider here, as integrating features over time should lead to more efficient search strategies.

Anderson et al. [4] emphasize the importance of memory mechanisms that support the construction of rich internal representations of environments in navigation agents. They argue that the nature of the internal representation is central to the study of embodied navigation.

Simple agents that are purely reactive and act on the sensory input at the current time step only work for simple tasks. Simple memory mechanisms, like the frame stacking used in [47] can model velocity and work well in certain environments where reactive action is sufficient. Once a task requires integration of features over longer time, more advanced memory is required.

Augmentations like recurrent update mechanisms add more potential. Hausknecht and Stone [31] investigate the effects of adding memory in the form of a recurrent step to DQN in order to tackle POMDPs. They use the same convolutional network as [47], but only use the most recent frame as input and replace the hidden layer with a recurrent LSTM layer. It is found that the agent is able to integrate information over time and achieves comparable performance to the original DQN agent. Several other works use a recurrent step to solve tasks that require memory, like [45] and [44]. General-purpose recurrent neural networks can in theory remember anything, but have practical issues like the the ones described in Section 2.1.3. Furthermore, they might not be easily trained for all tasks.

Specialized memory architectures can provide better performance for their intended tasks. Oh et al. [49] propose one such memory that can retain information over a fixed limited number of time steps. During each time step, the agent encodes its observation and shifts it into the memory. It also reads the memory using a soft attention [9] mechanism whose weights are determined by the encoded observation. This way, the agent can recall information from the past that is useful to the present. The authors evaluate this architecture in several visual navigation tasks, where image observations are encoded with a CNN. In their tests, their memory architecture performs better than LSTM layers, and generalizes better to unseen environments.

Several works propose the use of spatial memories rather than temporal ones, which the agent can address with its location and use to store structured information of its environment. Parisotto and Salakhutdinov [51] introduce a general-purpose spatial memory architecture. They assume that the environment can be discretized into a set of positions. The agent retains a feature map with one feature vector per position it can occupy in the environment. At each time step, the agent reads the feature map from the previous time step and writes information to the slot for its current position. This architecture is shown to outperform LSTM and the architecture in [49] in several navigation tasks which require remembering over many time steps. Several similar architectures have been proposed [33, 30, 17].

2.2.3 Generalization and Inductive Bias

In RL, generalization refers to an agents ability to act well in environments that have not been seen during training. This is vital if RL is to be applied to real-world problems where conditions are diverse and unpredictable. Agents have to be robust to such variations without being trained on them (*zero-shot* policy transfer). In the case of visual search, we want a system that can search in novel environments by generalizing from those seen during training.

While deep neural networks have proved to be effective function approximators for RL, they are also prone to *overfitting*. High-capacity models trained over a long time may memorize the distribution seen during training rather than general patterns. While studied in supervised learning, overfitting and generalization has generally been neglected in deep RL [39]. Training and evaluation stages are typically not separated. Instead, the final return on the training environments is used as a measure of agent performance. This results in agents that perform badly on environments that are only slightly different from those seen during training.

Zhang et al. [69] study overfitting and generalization in deep RL. With experiments, they show that RL agents are capable of memorizing training data, even when completely random. When the number of training samples exceeds the capacity of the agent, they overfit to them. When exposed to new but statistically similar environments during testing, test performance could vary significantly despite consistent training performance. The authors argue that good generalization requires the *inductive bias* of the algorithms to be compatible with the bias of the problems. The inductive bias refers to a priori algorithmic preferences, like neural network

architecture. When comparing MLPs with CNNs, they find that MLPs tend to be better at fitting the training data are worse at generalizing. When rewards are spatially invariant, CNNs generalize much better than MLPs. The authors advocate for carefully designed testing protocols for detecting overfitting. The effectiveness of stochastic-based evaluation depends on the properties of the task. Agents could still learn to overfit to random training data. For this reason, they recommend isolation of statistically tied training and test sets.

In a similar spirit, Cobbe et al. [21] construct distinct training and test sets to measure generalization in RL. They find that agents can overfit to surprisingly large training sets, and that deep convolutional architectures can improve generalization. Methods from supervised learning, like L2 regularization, dropout, data augmentation and batch normalization are also shown to aid with generalization.

Many current deep RL agents do not optimize the true objective that they are evaluated against, but rather a handcrafted objective that incorporates biases to simplify learning. Stronger biases can lead to faster learning, while weaker biases potentially lead to more general agents. Hessel et al. [34] investigate the trade-off between generality and performance from the perspective of inductive biases. Through experimentation with common reward sculpting techniques, they find that learned solutions are competitive with domain heuristics like handcrafted objectives. Learned solutions also seem to be better at generalizing to unseen domains. For this reason, they argue for avoiding biases determined with domain knowledge in future research.

Cobbe et al. [20] introduce a benchmark for sample efficiency and generalization in RL. They make use of procedural generalization, dependent on a random seed, to decide many parameters of the initial state of the environment. This forces agents to learn policies that are robust variation and avoid overfitting. To evaluate sample efficiency of agents in the benchmark, they train and test on the full distribution of states. To evaluate generalization, they limit the number of training samples and then test on held out levels. When an episode ends, a new sample is drawn from the training set. Agents may train for arbitrarily many time steps. The number of training samples required to generalize is dependent on the particulars and difficulty of the environment. The authors choose the training set size to be near the region when generalization begins to take effect. Empirically they find that larger model architectures improve both sample efficiency and generalization. Agents strongly overfit to small training sets and need many samples to generalize. Interestingly, training performance improves as the training set grows past a certain threshold. The authors attribute this to the implicit curriculum of the distribution of levels.

Kirk et al. [39] survey generalization in deep RL. They distinguish between methods that try to generalize from training to testing sets by increasing their similarity, and those that attempt to handle their differences. To handle differences between training and test sets, an agent's policy should rely only on features which will behave similarly in the training and testing context. If the similarities between training and testing contexts are known, they can be encoded as inductive bias for stronger generalization. When there is not specific inductive bias to encode, standard regularization can be used. Finally, when we can not rely on specific inductive biases or standard regularization, we have to rely on learning the invariances from training and hope that these generalize to testing.

They further propose a formalism for collections of problems called contextual CMDPs. A CMDP which is an MDP $\mathcal{M}$ (or POMDP) where the state can be decomposed into $s = \langle c, s' \rangle$, where $s \in \mathcal{S}$ is the underlying state and $c \in \mathcal{C}$ is the *context*. The context takes the role of a seed and determines the sample drawn from the underlying distribution of task instances. In a CMDP, separate train and test tasks can be defined by creating $\mathcal{C}_{\text{train}} \subseteq \mathcal{C}$ and $\mathcal{C}_{\text{test}} \subseteq \mathcal{C}$ such that $\mathcal{C}_{\text{train}} \cap \mathcal{C}_{\text{test}} = \varnothing$. It is argued that the only choice when evaluating generalization with procedurally generated environments is the training set size. Evaluation has to be performed on the full distribution of (held out) contexts.

2.2.4 Evaluation of Agents

A problem in state-of-the art deep RL is reproducibility. There is often non-determinism, both in the methods and environments used. Furthermore, many methods have intrinsic variance which can make published results difficult to interpret. In some cases, training an algorithm in an environment twice can give learning curves that do not fall within the same distribution [32]. This has meant that reproducing state-of-the-art deep RL results is difficult. A common solution is to report average results and variance across multiple different runs.

However, the sample inefficiency of current deep RL algorithms together with the rise in popularity of more challenging benchmarks has led to long training times. This has made it less feasible to measure performance over many training runs, which in turn has led to a shift to only evaluating a small number of runs ($N \leqslant 5$) per task. [2] This is problematic, as several analyses indicate that as many as $N = 20$ runs are required for statistically significant results [23, 22].

Henderson et al. [32] investigate reproducibility methods in deep RL empirically, focusing on policy gradient methods. They try to reproduce results in works with published code bases, and investigate how various modifications affect results.

They find that hyperparameters and the choice of network architecture for policy and value function approximation can affect performance significantly. Furthermore, rescaling rewards can have a large effect, although it is difficult to predict how. It is found that ReLU activations tend to perform best across environments and algorithms. For PPO, the use of large networks may require changing other hyperparameters like learning rate. Confidence bounds with sample bootstrapping is used to show that PPO is among the more stable algorithms.

For certain environments, learning curves can indicate successful optimization but the learned behavior may not be satisfactory. It is therefore important to not only show returns, but also demonstrations of the learned policy in action. Interestingly, implementation differences that are not reflected in publications can have a dramatic impact on performance. It is therefore necessary to enumerate implementation details and package code bases with publications. Performance of baseline experiments should also match original baseline publication code.

Finally, [32] make the point that more emphasis should be placed on applying RL algorithms to real-world tasks. It could be more useful to propose a set of tasks that an algorithm could be used for than to show performance on fictional tasks.

Anderson et al. [4] discuss problem statements and evaluation measures for embodied navigation agents, and make a set of recommendations. A navigation agent should be equipped with a special action that indicates that it has concluded the episode. The agent should be evaluated at the time this action is made, and not at some more favorable time step. Proximity to a goal should be measured using geodesic distance, the shortest distance in the environment. They recommend success weighted by (normalized inverse) path length (SPL) as the primary measure of navigation performance. With N test episodes, SPL is computed as

$$\text{SPL} = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)} \tag{2.9}$$

where S_i is a binary indicator of success, l_i is the shortest path distance from the agent's starting position to the goal, and p_i is the length of the path actually taken in the episode. SPL takes the quality of the solution into account: If 50% of test episode are successful and the agent takes the optimal path in all of them, its SPL is 0.5. By measuring SPL of human subjects, what is a good score can be calibrated.

3 Method

In this chapter, the method used to answer the research questions in Section 1.3 Section 3.1 formalizes the problem solved. Section 3.2 details the environment used to evaluate solutions. Section 3.3 describes the approach used to solve the problem with a learning agent. Section 3.4 describes the experiments conducted to answer research questions 2 and 3.

3.1 Problem Statement

We can now formally define the problem of searching for targets in unknown environments, adopting the CMDP formalism [39]. Let the task be a contextual POMDP $\mathcal{M}$ where the state includes the context c, which we refer to as the *seed*. The state includes the searched space $S \subset \mathbb{R}^d$ which we refer to as the *scene*. In the scene, there is a set of N targets $T = \{t_0, t_1, \dots t_N\}$ such that $t_i \in S$. At each time step, the agent perceives a subspace $V \subset S$ of the environment which we refer to as the *view*. If $T \cap V \neq \varnothing$ there are $|T \cap V|$ targets in view.

The actions $a \in \mathcal{A}$ transform the view into a different subset of the scene. With a final action, the agent can indicate that there is one or more target in the view. The observations $o \in \Omega$ are tuples $o = \langle x, p \rangle$. Here, $x \in \mathbb{R}^{3 \times w \times h}$ is an RGB image representing the current view, and $p \in S$ is the position of the agent which uniquely identifies the view. The goal of the agent is to select actions that bring each target into view and indicate that they are visible, while minimizing the number of total steps. There is no inherent reward $\mathcal{R}$ for this problem, so it has to be designed. Finally, seed c determines the initial view V, the location of the targets T, the initial position p_0 as well as the image observations x_t at each position p_t.

3.2 Environments

To train an test an agent for the problem, we use three different environments with similar observation spaces, action spaces and reward signal. Each environment have different characteristics that test the applicability of the evaluated approaches to different types search problems. In all environments, the appearance of the scenes and the location of targets are correlated to some degree. This means that the agent should be able to learn common characteristics of each environment and use those to search more efficiently.

As [20] and [44], we leverage procedural generation in all environments. This gives us control over the difficulty of the environments as well as the number of training and test

samples the agent is exposed to. A seed determines the appearance of the scene, the location of the targets and the initial position of the agent.

Each episode is terminated when all targets have been found, or after 1000 time steps Terminating episodes early this way is common to speed up training [50].

3.2.1 Observations, Actions and Reward

All three environments use the same observation space, action space and reward signal. The position and image observations are

$$o_t = \langle x_t, p_t \rangle, \text{ where} \tag{3.1}$$
$$x_t \in \mathbb{R}^{3\times 64\times 64}, \text{ and} \tag{3.2}$$
$$p_t = \langle p_{0,t}, p_{1,t} \rangle \in \{0, \ldots, H-1\} \times \{0, \ldots, W-1\} \tag{3.3}$$

All image observations are 64×64 RGB images. The agent moves in a $H \times W$ grid, and we assume the presence of some oracle that provides the agent with its position.

The action space is the same in all environments:

$$a_t \in \{\texttt{UP}, \texttt{DOWN}, \texttt{LEFT}, \texttt{RIGHT}, \texttt{INDICATE}\}, \tag{3.4}$$

where `UP`, `DOWN`, `LEFT`, and `RIGHT` move the view one step in each direction. The final action, `INDICATE`, is used to indicate that a target is in view.

The reward signal should be designed so that the agent learns a policy that achieves the goal of finding all targets with a minimal number of actions. We use the following reward signal:

$$r_t = h - 0.01 + 0.005d + 0.005e \tag{3.5}$$

Here, $h = |T \cap V|$ if $a_t = \texttt{INDICATE}$, and $h = 0$ otherwise. This term is equal to the number of targets that were found at this time step. The constant penalty of -0.01 ensures that the agent is rewarded for quick episode completion. The term $d = 1$ if p_t is closer to the nearest target than p_{t-1}, otherwise $d = 0$. Similarly, $e = 1$ if p_t has not been explored previously and $e = 0$ otherwise.

Through experimentation, we find that a larger time penalty than -0.01 dominates the reward for finding targets. This is potentially related to the episode time – with a time penalty thats too large, the reward h for finding targets has a relatively small impact on episode return for long episodes.

The two next terms are bonus rewards, intended to speed up learning by further encouraging desired behavior. Actions that move the agent towards the nearest target and move the view to previously unseen regions are desirable. Importantly, the sum of these bonuses is not larger than the magnitude of the time penalty. This is to ensure that exploration and moving towards targets does not seem more important to the agent than finishing the episode quickly. Therefore, the bonuses may guide the agent towards finding targets but do not cause it to steer away from the underlying goal.

3.2.2 Gaussian Environment

The first environment is the simplest environment, where the correlation between scene appearance and target probability is clear. During each episode reset, a 1024×1024 RGB image is generated conditioned on the seed. The image contains three blue spots, whose intensity is highest towards their center and wears off radially outwards. The intensity wears off as an (approximate) Gaussian function. The sum of the intensity in the blue channel in the image is used as a probability density function to select the location of three targets. Targets are

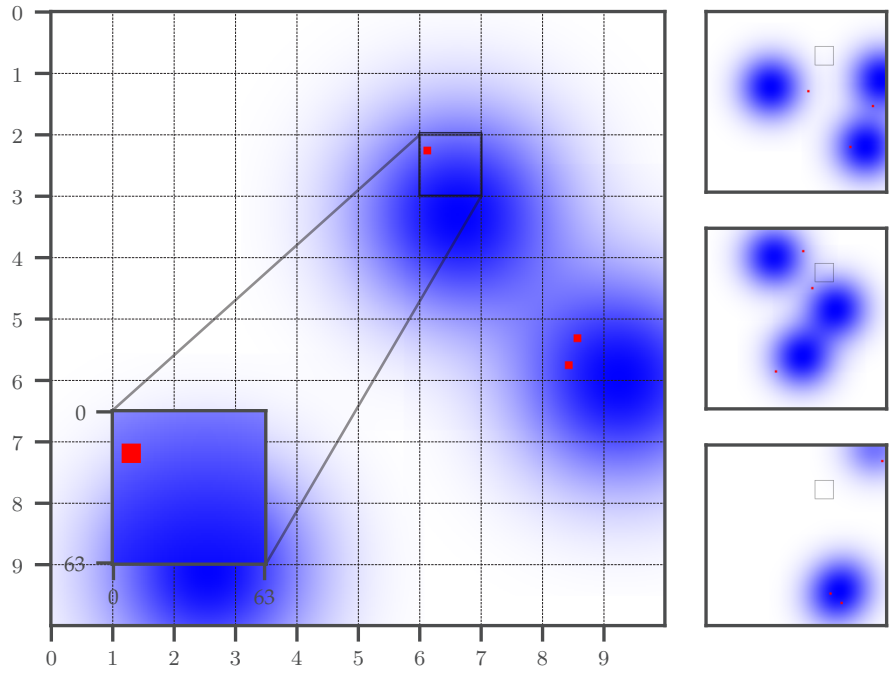

Figure 3.1: Four samples of the first environment. Agent observes 64×64 image at position $(2, 6)$. There are three blue spots and three red targets in each scene. Targets are more likely where the intensity of the spots is high.

characterized by red 8×8 squares. The higher the intensity in the blue channel, the more likely that there is a target there.

The image is divided into a grid of 10×10 steps, one per position p_t. The image x_t is the 64×64 sub-image at the current position in the grid. Each moving action translates the agent one step in corresponding cardinal direction in the grid.

The idea with this environment is to test whether an agent is able to utilize scene characteristics to search quicker. An efficient searcher in this environment should prioritize locations where the intensity in the blue channel is high. It should be able to use the gradient of the underlying Gaussian function to move towards locations with higher target probabilities, while avoiding revisiting locations and intelligently planning its search path.

3.2.3 Terrain Environment

The second environment is similar to the first one, but intended to simulate a search scenario in realistic terrain. Actions and observations behave the same as in the first environment. At the start of each episode, a 1024×1024 height map is generated using gradient noise. The height map is used to determine the color of an image of the same size. Lower heights are filled with blue ocean, and higher areas with green grass and brown mountains. Three red targets are located with uniform probability along the shores, between mountains and water. Green trees are also scattered around each scene, whose positions are sampled from the same distribution as that of the targets.

While this environment is similar to the first environment, it is less clear how to search efficiently in it. There is higher variance between scene samples. It is also less clear how the scene appearance is correlated to the probability of targets. It is desirable that a searching agent should learn to not search oceans and mountains. Instead it should prioritize searching along the edges of land masses. For some scene samples, there are multiple islands with

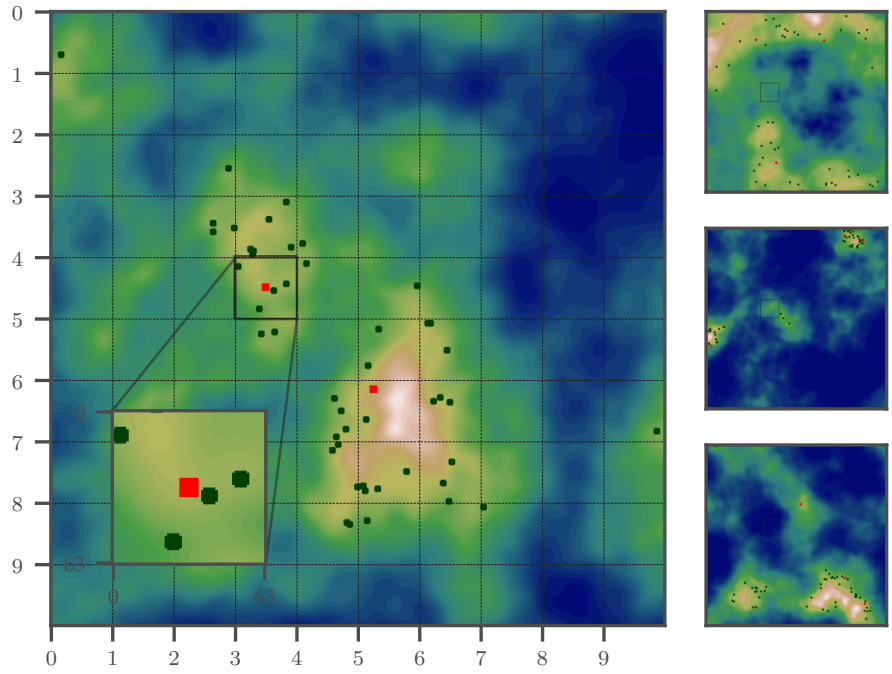

Figure 3.2: Four samples of the second environment. Terrain seen from above with red targets scattered and green trees scattered along shores. Agent observes 64×64 image at position $(4, 3)$

small patches of land between them. These patches could be used to quickly prioritize land while avoiding water. One can draw parallels to search-and-rescue scenarios with UAVs or fire detection.

3.2.4 Camera Environment

The third environment is a three-dimensional version of the second one. The height map is turned into a three-dimensional mesh, and the agent is placed at its center. Targets are, as before, placed along island edges.

The agent observes the scene through a pan-tilt perspective camera. Moving actions rotate the camera instead of translating it. The `LEFT` and `RIGHT` actions control the yaw of the camera, while `DOWN` and `UP` control its pitch.

The yaw is divided into 20 steps between 0 and 360 degrees. The yaw angle wraps around, so that the agent can look around freely. Similarly, the pitch is divided into 10 steps between 0 (straight forward) and -90 degrees (straight down), but without wrapping around. This means that the camera can take 200 different positions.

Targets are always visible from at least one camera position. They may be visible from multiple positions, and the agent is expected to use the `INDICATE` action only when a target is as close to the center of the view image as possible. This environment is intended to model more realistic scenarios where the image is more difficult to interpret. In some scenarios it can be important that objects are not only localized, but localized accurately.

3.3 Approach

To design an agent that effectively solves the task, we draw inspiration from several previous works and adapt them to better suit this particular task. The final agent should be able

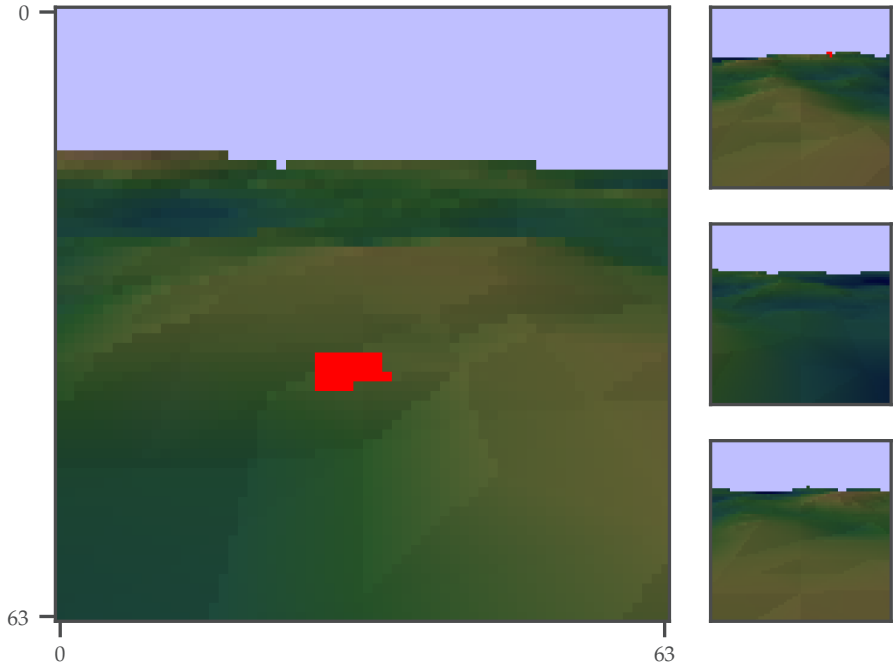

Figure 3.3: Four samples from the camera environment. Terrain seen from a pan-tilt camera. The pan and tilt of the camera can be adjusted to move the view around.

to recognize targets, regardless of where they appear in view. It is likely important for the agent to have access to its location, especially for large search spaces. As environments are procedurally generated, a partial image observation of the scene is not sufficient to ground the agent in it. It should also be able to integrate features over time in order to remember which locations have been visited. Remembering visual features of of these locations may also be of importance, as it can provide clues for what is in their proximity.

Due to the advantages described in Section 2.1.4, we limit our approaches to policy gradient methods. Specifically, we employ an actor-critic approach that estimates a policy and value function with a multi-headed neural network. The neural network architecture should reflect the aforementioned requirements.

3.3.1 Architecture

We design our neural network architecture as follows: A CNN takes the observed image x_t and encodes it into a latent representation h_t. This allows the agent to extract translation invariant features from observed images. The latent representation, as well as the current position of the agent p_t is used as input to an RNN. Feeding both an image representation and the position of the agent to a recurrent step lets the agent remember visited locations and their appearance. The output of the recurrent step is in turn is connected to an actor MLP head and a critic MLP head, which approximate the policy π and value function v respectively. The architecture of the neural network is presented in Figure 3.3.1.

For the CNN, we use the same architecture as [47]. The input image $x_t \in \mathbb{R}^{3\times64\times64}$ is fed through three convolutional layers: the first layer convolves 32 filters of size 8×8 with stride 4, the second convolves 64 filters of size 4×4 with stride 2, and the third layer convolves 64 filters of size 3×3 with stride 1. This is followed by a final fully connected hidden layer with 512 outputs for the latent representation $h_t \in \mathbb{R}^{512}$.

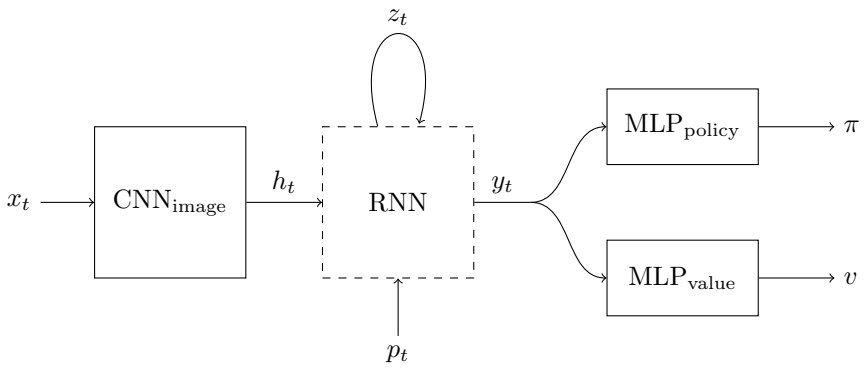

Figure 3.4: The neural network architecture used for estimating the policy and value functions from image and position observations.

Both the policy and value network are fully connected networks. The value network has one output for the value estimate. The policy network has 5 outputs, the logits for each action. Applying the softmax operation on this output gives the final action probabilities. All network layers have ReLU activation functions, as suggested by [32].

Two RNN architectures are compared: one temporal, and one spatial. While a temporal memory can retain location and appearance information over time, it is not cleared how this information is stored and whether it can be utilized properly. We hypothesize that agents using temporal memories may struggle with learning good policies in large search spaces and those that require scene understanding, such as reasoning over previously visited locations. Several results from works in embodied visual navigation indicate that spatial memories can give better results than temporal ones [51, 33, 30, 17]. Taking inspiration from this, we investigate whether a spatial memory can be more useful than a temporal one when searching for targets.

Temporal Memory

The temporal memory is a single LSTM [35] layer, as proposed by [31] and used in [44], [43], and [30]. As input to the LSTM, we use the latent image representation h_t concatenated with one-hot encodings of each dimension of p_t. The LSTM layer has 128 hidden cells so that y_t has 128 dimensions. Finally, z_t contains the hidden and cell states.

Spatial Memory

The spatial memory is composed of a readable and writable feature map. This is a simplified version of the architecture proposed in [51], similar in spirit to the architectures in [33], [30] and [17]. It is defined by the following set of operations:

$$\rho_t = \text{CNN}_{\text{read}}(z_t) \tag{3.6}$$

$$\omega_t = \text{MLP}_{\text{write}}(\left[h_t, \rho_t, z_t^{(p_t)}\right]) \tag{3.7}$$

$$q_t = \text{MLP}_{\text{position}}([\text{onehot}(p_{0,t}), \text{onehot}(p_{1,t})]) \tag{3.8}$$

$$y_t = [\rho_t, \omega_t, q_t] \tag{3.9}$$

$$z_{t+1}^{(p')} = \begin{cases} \omega_t & \text{if } p' = p_t \\ z_t^{(p')} & \text{if } p' \neq p_t \end{cases} \tag{3.10}$$

Table 3.1: PPO hyperparameters used during training.

Parameter	Value
total time steps	25×10^6
discount factor γ	.999
advantage factor λ	.95
parallel environments N	64
time steps per rollout T	256
epochs per rollout K	3
mini-batches size M	2048
value loss coefficient c_1	0.5
entropy bonus coefficient c_2	0.01
clip range ϵ	.2
learning rate α	5×10^{-4}
reward normalization	yes
learning rate schedule	linear

Here, $z_t \in \mathbb{R}^{64\times10\times10}$ is a feature map with one 64-dimensional feature vector per possible position p_t. At each time step, the agent reads from this map using a CNN with three convolutional layers. Each layer has 32 filters of size 3×3, and uses a padding of 1. A final fully connected layer produces a 64-dimensional read vector ρ_t. An MLP takes the read vector concatenated with h_t and the feature vector stored in the map at the current position $z_t^{(p_t)}$, and produces a 64-dimensional write vector ω_t. Another MLP takes a one-hot encoding of the position as input and outputs a 64-dimensional vector q_t. The read vector ρ_t, the write vector ω_t, and the position vector q_t are concatenated and used as input y_t to the value and policy networks. This means that they can make use of visual features of previously explored locations, spatial relationships between these features, and the current position of the agent. Finally, the feature map is updated so that z_{t+1} contains the write vector ω_t at position p_t.

3.3.2 Training

We train both agents with PPO [56], as described in Section 2.1.4. Early experiments show that PPO gives good results, stable learning curves and good sample efficiency, which is in line with results reported by [6]. Furthermore, we use similar hyperparameters to [20]. These are presented in Table 3.1. Many parallel environments seem to both speed up and stabilize training. As suggested by [6], we initialize the policy output weights so that their mean is 0 and their standard deviation is low (10^{-2}).

We normalize the reward using a moving average. This is done to limit the scale of the error derivatives. Early experiments show that not normalizing the reward destabilized learning. Similar results have been reported by [6] and [46]. The Adam optimizer [38] is used in all experiments. Finally, we decay the learning rate linearly so that it is zero at the final time step. We find that this helps the agent with finding a better local optimum.

3.4 Experiments

We conduct three different experiments to evaluate our approaches and answer the research questions in 1.3. Following the recommendations of [32], [22] and [2], we report mean and standard deviation across a handful of seeds.

3.4.1 Search Performance and Behavior

To evaluate the quality of the learned policy, we compare our approaches to a set of baseline policies: one random, one greedy, and one exhaustive. The random policy (Algorithm 3.4.1), selects random moving actions. The greedy policy (Algorithm 3.4.1) keeps track of visited positions, and greedily selects actions that explore new positions. If no such actions exist, it selects a random moving action. The exhaustive policy (Algorithm 3.4.1) behaves similarly to the second one. Instead of sampling exploring actions randomly, it selects exploring actions from a predetermined order. In rectangular search spaces without obstacles, this policy leads to search paths that cover the whole search space with a small amount number of revisited positions.

All baselines automatically indicate when a target is visible. We feel that offloading the detection from the baselines is reasonable. Having the baselines randomly indicate that targets are visible would lead to excessively long search paths in moderately large search spaces. By giving this advantage to the baselines, the emphasis of the comparison is put on the search path planning.

Algorithm 2 Random Baseline Policy

if there is a target at p_t **then**
  $a_t \leftarrow$ `INDICATE`
else
  $\mathcal{A}_{\text{move}} \leftarrow \{\texttt{UP}, \texttt{RIGHT}, \texttt{DOWN}, \texttt{LEFT}\}$
  $a_t \leftarrow$ sample from $\mathcal{A}_{\text{move}}$
end if

Algorithm 3 Greedy Baseline Policy

if there is a target at p_t **then**
  $a_t \leftarrow$ `INDICATE`
else
  $\mathcal{V} \leftarrow \{p_t, p_{t-1}, \dots p_0\}$
  $\mathcal{A}_{\text{move}} \leftarrow \{\texttt{UP}, \texttt{RIGHT}, \texttt{DOWN}, \texttt{LEFT}\}$
  $\mathcal{A}_{\text{explore}} \leftarrow \{a | a \in \mathcal{A}_{\text{move}}$ moves the agent to an unvisited location $p \notin \mathcal{V}\}$
  if $\mathcal{A}_{\text{explore}} \neq \varnothing$ **then**
    $a_t \leftarrow$ sample from $\mathcal{A}_{\text{explore}}$
  else
    $a_t \leftarrow$ sample from $\mathcal{A}_{\text{move}}$
  end if
end if

Algorithm 4 Exhaustive Baseline Policy

if there is a target at p_t **then**
  $a_t \leftarrow$ `INDICATE`
else
  $\mathcal{V} \leftarrow \{p_t, p_{t-1}, \dots p_0\}$
  $\mathcal{A}_{\text{move}} \leftarrow \{\texttt{UP}, \texttt{RIGHT}, \texttt{DOWN}, \texttt{LEFT}\}$
  $\mathcal{A}_{\text{explore}} \leftarrow \{a | a \in \mathcal{A}_{\text{move}}$ moves the agent to an unvisited location $p \notin \mathcal{V}\}$
  if $\mathcal{A}_{\text{explore}} \neq \varnothing$ **then**
    $a_t \leftarrow$ the first a in $\langle \texttt{UP}, \texttt{RIGHT}, \texttt{DOWN}, \texttt{LEFT} \rangle$ such that $a \in \mathcal{A}_{\text{explore}}$
  else
    $a_t \leftarrow$ sample from $\mathcal{A}_{\text{move}}$
  end if
end if

For each of the three environments, we train our approaches on the full distribution of samples. Using a fixed set of 100 held out test samples, we measure the search path length and success rate of each approach and baseline. When measuring the average search path length, we only include successful searches. Finally, we investigate how the search performance of our approach compares to that of a human with prior knowledge of each environment.

We also report the SPL metric [4] for all agents, where the shortest path length is the optimal travel distance between the targets and the initial position of the agent. The distance between two points is computed as the minimal number of actions to transform the view between the two. Although finding the optimal path is not realistic in partially observable environments, SPL can still be useful when compared to that of a human. For each metric and agent, we report both the mean and standard deviation across three runs.

3.4.2 Size of Search Space

The number of steps required to locate all targets in a scene is dependent on the size of the search space. In small search spaces, the difference in performance between an intelligent searcher and a less intelligent one may be difficult to quantify. Furthermore, a large scene is more difficult to search intelligently - more capacity is needed to remember visited locations and plan future steps.

To investigate how well our approach scales to different search space sizes, we train our approaches on three variants of the gaussian environment. The scene image is scaled up so that there are 10×10, 15×15 and 20×20 possible camera positions in each environment. We train and test both approaches on the full distribution of samples from all three search space sizes respectively. By varying the search space size in each environment, we can get a feel for how the size affects training times and the quality of the learned policies. For each agent and search space size, we report how the average length and success rate changes while learning.

3.4.3 Number of Training Samples

In realistic scenarios, training agents on an unlimited number of samples is not possible. It is more likely that there is a limited set of samples for an agent to learn from. For this reason, it is interesting to quantify how the number of training samples affects generalization to the full distribution.

We do this by training our two agents on 500, 1000, 5000, 10000 and an unlimited number samples of the terrain environment. Both agents are tested on held out samples from the full distribution of samples, as suggested by [20]. This should illustrate how many samples are needed to generalize, and if overfitting is an issue. For each agent and training set size, we report how the average length and success rate changes while learning.

3.5 Implementation

The environment is implemented with using the Gym [14] interface. The agent is implemented and RL algorithms are implemented with PyTorch [52] for automatic differentiation of the computation graphs. PPO [56] was implemented following the official implementation by OpenAI, and verified by testing on the benchmarks used in the original paper. All experiments are conducted on an Intel Core i9-10900X CPU and an NVIDIA GeForce RTX 2080 Ti GPU. The source code for environments, models and algorithms is available at `https://gitlab.liu.se/osklu414/tqdt33-masters-thesis`.

4 Results

This chapter presents the results for each of the experiments described in Section 3.4. In addition to the results here, we present learning curves for all environments in Appendix A and some search path examples for our approaches and baselines in Appendix B.

4.1 Search Performance and Behavior

Table 4.1 shows the average search path length, success rate and SPL metric on a fixed set of 100 levels from each environment. The average search path length is computed from episodes in which the agents successfully finds all targets. These metrics are presented for our two approaches trained on the full distribution of environments, as well as for the four baselines and human searchers.

Overall, our two approaches and human searchers achieve similar scores. Our spatial memory approach achieves the most competitive search path lengths and SPL scores in the gaussian and terrain environments. Our temporal memory agent select search paths that are on average longer than the paths chosen by our spatial memory approach and human searchers. In the camera environment, the temporal memory agent is better both in average length and SPL score. Interestingly, the temporal memory agent achieves SPL scores comparable to that of the spatial memory agent despite choosing much longer path lengths.

Human searchers seemed to be successful in utilizing using environment cues to guide search, but frequently forgot which positions had been visited. In the camera environment, human searchers found it difficult to identify when targets should be indicated, as the discretization of the search space did not always align targets closely to the center of the screen. This is more of a side-effect of the implementation than an indication of poor search performance.

The random baseline policy achieves the worst SPL, success rate and average search path length in all three environments. The greedy baseline policy is closer to the remaining agents. The exhaustive baseline policy achieves an average search path length that is lower than our temporal memory approach. Its SPL score, however, is only higher than the those of the other two baselines.

Table 4.1: SPL, success rate and average search path length on successful episodes from three runs on a fixed set of a 100 samples from each environment.

Gaussian environment

Agent	SPL	Success	Length
random	0.06 ± 0.01	0.92 ± 0.06	369.07 ± 24.93
greedy	0.17 ± 0.00	1.00 ± 0.00	147.12 ± 2.38
exhaustive	0.21 ± 0.00	1.00 ± 0.00	83.37 ± 2.88
human	0.23 ± 0.03	1.00 ± 0.00	80.97 ± 13.49
lstm	0.25 ± 0.02	0.99 ± 0.01	108.38 ± 10.44
map	0.29 ± 0.02	0.99 ± 0.01	72.16 ± 5.97

Terrain environment

Agent	SPL	Success	Length
random	0.06 ± 0.01	0.89 ± 0.04	366.05 ± 26.96
greedy	0.17 ± 0.01	1.00 ± 0.00	141.01 ± 2.31
exhaustive	0.22 ± 0.00	1.00 ± 0.00	84.11 ± 0.84
human	0.26 ± 0.02	1.00 ± 0.00	76.73 ± 5.33
lstm	0.25 ± 0.02	1.00 ± 0.01	103.76 ± 11.69
map	0.27 ± 0.01	1.00 ± 0.00	79.60 ± 6.88

Camera environment

Agent	SPL	Success	Length
random	0.04 ± 0.00	0.62 ± 0.03	545.09 ± 56.25
greedy	0.12 ± 0.01	0.97 ± 0.01	255.60 ± 10.44
exhaustive	0.37 ± 0.00	1.00 ± 0.00	67.03 ± 0.00
human	0.68 ± 0.08	1.00 ± 0.00	38.10 ± 5.72
lstm	0.70 ± 0.02	1.00 ± 0.00	42.36 ± 2.05
map	0.66 ± 0.03	1.00 ± 0.00	42.90 ± 1.73

4.2 Size of Search Space

The results of the search space experiments in the gaussian environment are presented in Figure 4.2. Results were collected across four different runs, and the plots show the mean and standard deviation. For the search space of 10×10, both architectures initially improve their policy quickly. Past a certain time step, they keep improving at a reduced pace. At the end of training, the spatial memory architecture has reached a policy that seems to find targets quicker than the temporal memory. Both seem to be able to find the all three targets in every episode.

For the larger search space sizes with 15×15 and 20×20 the difference between the two architectures is greater. While the spatial memory seems to consistently find targets in a number of steps that is comparable to the number of positions in the search space, the search paths of the agent with the temporal memory are substantially longer. Furthermore, the variance across runs increases with the search space size.

4.3 Number of Training Samples

Figure 4.3 shows how the average length and success rate in the terrain environment is affected by the number of samples seen during training. These metrics are presented for the limited training set and unlimited testing set respectively.

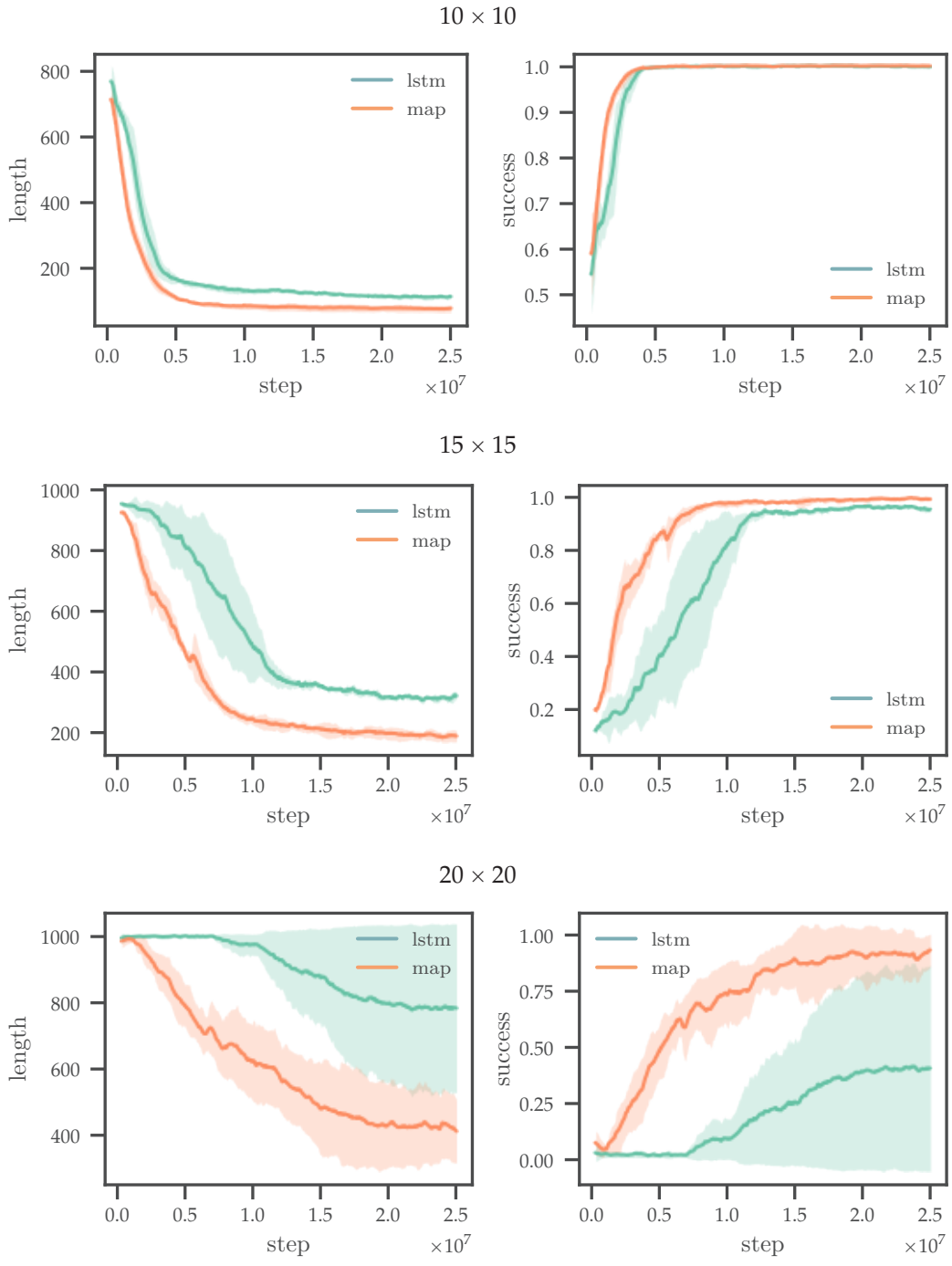

Figure 4.1: Episode length and success rate curves during training for three different search space sizes. Mean and standard deviation across 4 runs.

It seems like both architectures can overfit to training sets of as many as 1000 samples, For smaller training set sizes, search path lengths on the test get get gradually worse past a certain time step. Interestingly the LSTM architecture seems to overfit more severely to small training sets. Its performance on the test set decreases substantially past a certain time step. The spatial memory architecture does not As many as 10000 training samples are needed to generalize to the full distribution of scenes and achieve the same performance on the training and test sets.

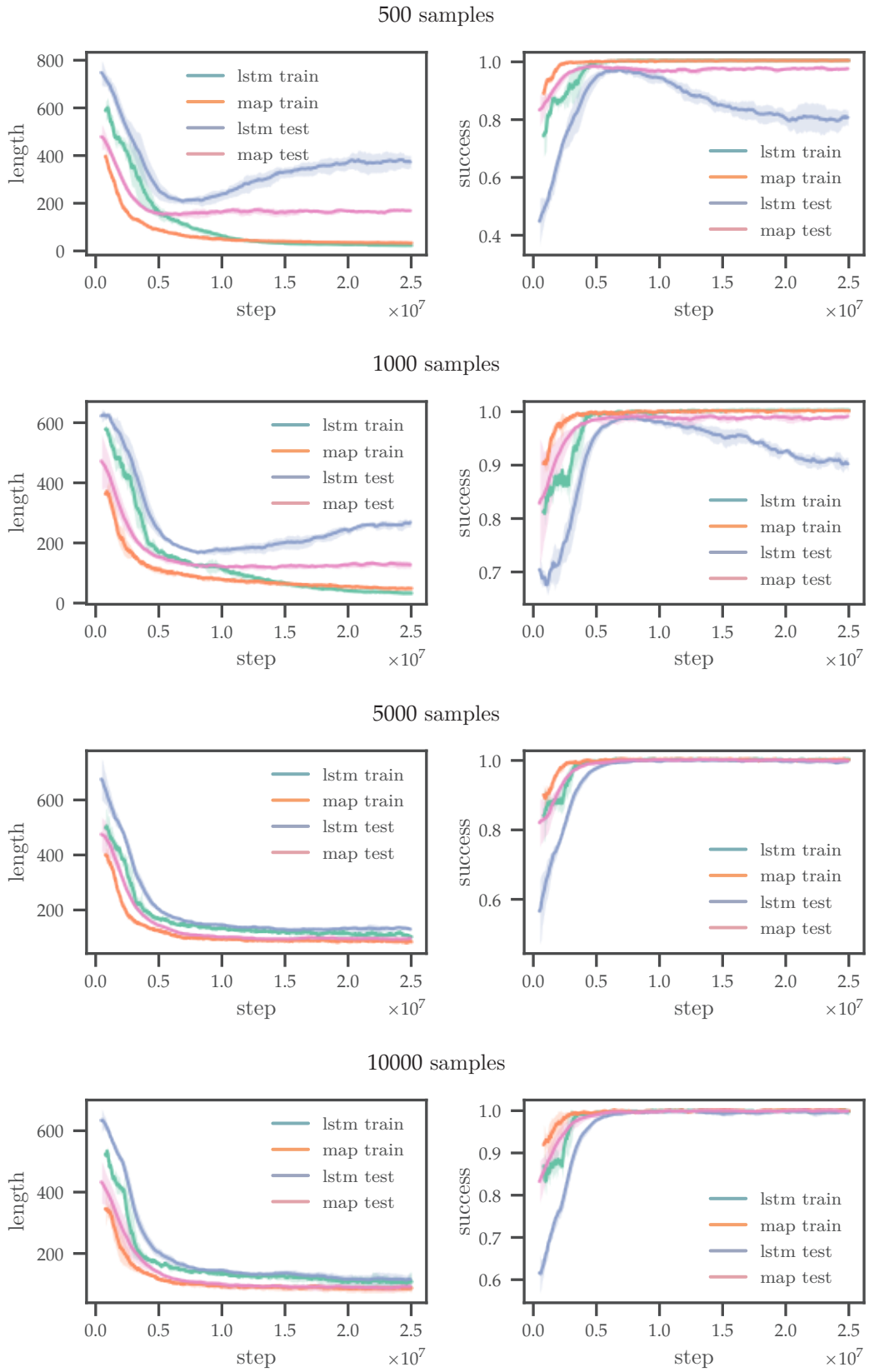

Figure 4.2: Episode length and success rate curves for different training set sizes in terrain environment. Mean and standard deviation across 3 seeds.

5 Discussion

In this chapter, we reflect the obtained results and critically discuss and evaluate the chosen method. We also discuss the work from a broader perspective (Section 5.3)

5.1 Results

5.1.1 Reflection on Search Performance

To evaluate the search quality, we first consider the at the average search path length of each agent. It is useful to compare the path length to the number of points in each search space. The gaussian and terrain environments contain $10 \times 10 = 100$ searchable positions, and the camera environment contains $10 \times 20 = 200$ searchable positions. A search path length above this suggests many revisited positions. The exhaustive baseline policy provides a good measure for what sort of path lengths can be achieved when only avoiding revisiting positions. Both the random and greedy baseline policies select search paths that are on average much longer than the number of positions, suggesting several redundant steps. This is further reinforced by the sample search paths in Figures B, B and B.

Notably, our temporal memory agent chooses search paths that are substantially longer than those selected by our spatial memory agent and human searchers. This is especially true for the gaussian and terrain environments, which suggests several redundant steps. Our spatial memory agent achieves better average lengths in these two environments.

We see that search paths are particularly short in the camera environment, despite its larger search space. This has two reasons: First, targets may be visible in the periphery from a large set of camera positions. Once targets are visible, it is simpler for an agent to select actions that center them. Second, targets are not distributed uniformly across many samples of the camera environment. They are most commonly found near the horizon. In the gaussian and terrain environments, where the distribution is approximately uniform over many samples, the search path lengths are closer to the number of positions in the search space.

Our temporal memory agent performs very well in the camera environment. We suspect that this is due to the fact that less emphasis is placed on remembering over many time steps. Search in this environment can be more reactive. When inspecting the search patterns used by our temporal memory agent in this environment, we see that it has learned to alternate between scanning across the horizon and locating targets in its periphery. The baselines are

unable to utilize this imbalance in target distributions. The exhaustive policy is an exception, where the deterministic ordering of actions happened to favor that distribution.

By looking at some sample search paths, we see that our two approaches both seem to utilize scene appearance to guide their search. However, they also tend to revisit positions, and it is not clear that this is good. Figures B and B illustrate this well. Some of these revisits could be attributed to the stochastic policy. Making the policy completely deterministic by selecting the action with the highest probability is not a solution, as it tends to lead to dead states or loops where the agent does not progress.

Worth noting is that revisiting positions is not always the wrong choice. One such example is when the current position of the agent is directly between two targets. The optimal path to find both of these is to first go to any one of them, and then to go back towards the other one by revisiting positions.

Next, we consider the SPL score of each agent. When averaged over many episodes, the SPL score should be a better measure of search path quality than average length, as it also accounts for how close each path is to the optimal one. From the definition of the SPL metric in Equation 2.9, we that it favors search paths whose length are close to the shortest one in each sample. Finding the shortest path in a search scenario such as the one we consider here is not feasible. However, it still serves as a useful normalization method as it is an indication of how difficult each sample is to solve.

Both our two approaches and human searchers achieve similar SPL scores in all three environments. While the exhaustive baseline policy achieves average path lengths that are close to those of our two approaches and human searchers, its lower SPL score suggests that it is less successful in finding close to optimal solutions. In all three environments, one of our approaches achieve a higher SPL score, although we would not consider this significant due to the large variance.

Under these metrics, both approaches achieve comparable performance to that of a human searcher. Overall, it would seem like our spatial memory approach is most suited where large search spaces have to be explored strategically, while our temporal memory approach is sufficient for search problems where search spaces can be explored reactively.

5.1.2 Scalability To Different Search Space Sizes

In Section 4.2 we saw that both of our approaches achieve comparable performance on the 10×10 search space. Both quickly converge towards a high success rate and reasonable search path lengths. The spatial memory is consistently capable of finding all targets in less steps than the number of searchable positions, and the temporal memory is not far behind.

For larger search spaces, the slight difference is accentuated. The temporal memory agent converges to search paths that are longer than needed, suggesting that it is not capable of remembering visited locations. In the 20×20 search space, it seems to have converged to path lengths of around 800, and has a success rate of less than 50%. By the end of training, the search paths of the spatial memory agent are near the total number of positions and it has a success rate of around 90%. Furthermore, the training does not seem to have fully converged.

It seems reasonable to state that the spatial memory scales better to large search spaces in terms of performance, when compared to the temporal memory. Searching efficiently in large search spaces places extra emphasis on remembering what has already been explored. From our results, it would seem like an LSTM struggles with remembering such precise information over many time steps. Similar results where presented by [49].

An hypothesis is that our temporal memory agent would handle large state spaces better with more parameters. We experimented with stacking multiple LSTM layers, but found that it led to overfitting and unstable learning, even with dropout. Further analysis is needed to see if such augmentations can make it more comparable to the spatial memory in terms of performance.

Even though we have investigated how our approaches scale to larger search spaces, it should be noted that realistic search spaces are often considerably larger. This poses a problem for both of our approaches, as the number of trainable parameters scales with the search space size in both of them. One solution is to keep search spaces small: Large (and continuous) spaces could be discretized into smaller ones at the cost of precision. Limit the need for memory could be a viable solution for certain search problems.. In the case of the temporal memory, it might be sufficient to only remember recent interactions. Similarly, in the case of the spatial memory, a possibility is to use an ego-centric recurrent feature map and only remember proximous interactions [51].

5.1.3 Generalizing from Limited Training Sets

From the results in Section 4.3, we see that both of our approaches can overfit to small training sets. This is in line with what is reported by [20], [21] and [69]. Past a certain number of interactions with small a small training set, performance on a held out test set decreases. This effect is well known in supervised learning, but important to keep in mind for reinforcement learning. If any sort of generalization is expected from our agents, they should be tested on held out samples.

Interestingly, our spatial memory agent seems to be better at generalizing from a limited number of samples – For 500 and 1000 samples, its performance on the test set does not decrease as severely as our temporal memory agent. Furthermore, the test performance of our spatial memory agent is strictly better than that of our temporal memory agent in all training set sizes. This suggests that the spatial memory provides a more suitable inductive bias for the considered problem.

It should be stated that these numbers are not representative of all search problems. The number of samples needed to generalize to other search tasks may be different than for the terrain environment we used in Section 4.3. While the terrain environment exhibits more subtle patterns than the gaussian environment, it is free from noise and is relatively low in variance compared to realistic scenarios. However, the results do offer insights into how these two architectures compare when it comes to generalization from a limited number of samples.

5.2 Method

5.2.1 Observations and Detections

The observations in the three environments in this work contain the position and an image of the current view. It seems reasonable to assume that the position can be provided to the agent. In many realistic scenarios it is possible to determine the global position of an agent (GPS, pan/tilt, etc.). If how each action moves the agent is well-defined, the relative position can be used instead of the absolute one.

In many cases, a 64×64 images are not sufficient to convey the full meaning of a scene. While we explicitly delimit ourselves from difficult detection problems in this work, detection is likely to be difficult in most real-world search scenarios. Further investigations are needed to evaluate how viable our approach is for more realistic scenes, where variance is higher and it is more difficult to interpret image observations.

Related to the detection problem is our choice of including an indication action. It could be argued that explicitly indicating when targets are in view is redundant. Indications could instead happen implicitly when targets are in view. This is true if the focus is only on how environment cues can be used to guide visual attention during search. Including explicit indication actions is in our view more interesting, as it requires some form of intentionality and is necessary for actually locating targets.

Furthermore, the indication action has several interesting applications. One can imagine a task in which target detection is done through some external object detection that is expensive

to run. In such a case, the indication could be used by the agent to convey that it is likely that a target is in view, and it is worth spending resources on running the detection process.

5.2.2 Actions and Search Space Dimensionality

The action space we chose in this work contains four different moving actions that change the position of the agent in two dimensions. This is reasonable for several real-world search tasks such as search and rescue with a UAV, where the actions correspond to translations in each cardinal direction. Another example is surveillance with a pan-tilt-camera, where the actions could correspond to pitch and yaw rotations.

Many vision systems have more degrees of freedom. Camera sensors often have zoom functionality, which in practice requires additional actions for zooming in and out, and adds an extra dimension to the search space. Similarly, if an agent is able to both move and look around we get additional dimensions to consider.

Nothing suggests that our two approaches would be unable to handle more than two dimensions. The temporal memory simply has to encode an additional component of the agent's position. The spatial memory can store a higher dimensional feature map. Its read operation can be modified to use convolutions in higher dimensions, as noted by [51].

5.2.3 Implications of Reward Signal

Finally, our reward signal assumes certain knowledge of the scenes: The environment (or the provider of the reward signal) has to keep track of visited positions. It also has to know where targets are located – even when they are not visible. While the reward is not needed when acting according to the policy, it is needed when learning. This has implications for what kind of environments can be used for learning. One can imagine a scenario where the locations of targets are not known, but a black-box system is responsible for detecting them and providing positive feedback to the RL agent. In this case, the bonus for moving towards targets is not computable.

In the gaussian environment, we found that the additional reward bonuses where required for convergence to good policies. One reason for this could be that it is difficult to interpret compared to the other environment. With a small bonus for desired behavior, reinforcements are more frequent and may aid with picking up subtle patterns. However, a specialized reward like ours risks introducing bias that inhibit learning by leading to poor local optima. Several works argue for minimizing such biases, as learning systems can sometimes find better solutions without them [34]. There is often a trade-off between strong bias/fast learning and weak bias/more general agents. For our set of environments, this particular reward seems to work well, but this might not be the case for search in other types of scenes.

5.2.4 Reinforcement Learning for Visual Search

It is worth considering whether using a learning agent is suitable for visual search problems. Several of the works which we cover in Section 2.2 utilize other solution methods with apparent success. Such methods can sometimes be probably correct and efficient. It is not obvious that RL is a good choice, and it has even been shown experimentally to struggle with generalizing to unseen samples in certain navigation scenarios [24]. One could imagine that it is possible to compute an optimal strategy for certain environments.

The perspective we take with this work is that non-learning solution methods may offer these guarantees, but designing such approaches for every search problem is laborious. The characteristics of environments can vary considerably which may drastically affect how a manual approach is implemented. Interpreting and utilizing patterns in arbitrary scenes may even be difficult for us humans. If we cannot understand patterns ourselves, we can not expect to

communicate how to search to machines. Furthermore, there have been several examples of learning systems surpassing humans in specific tasks [58, 63].

These two points alone are strong arguments for exploring whether learning systems can be good searchers. We would argue that RL is a suitable framework for implementing such learning agents. Our approach seems to be able to utilize environment appearance to search more efficiently. It is less clear if their behavior is close to optimality or not.

5.2.5 Replicability, Reliability and Validity

As discussed in Section 2.2.4, reproducibility is a problem in current deep RL research. This comes with the risk of stagnating the field by inhibiting correct interpretation of results [32]. In this work, we have taken certain precautions to avoid such problems.

For each experiment, we have collected results across at least 3 runs. While several sources state that as many as 10 runs are needed for significant results [22, 2], such undertakings are not feasible without sufficient compute resources. Our hope is that 3 runs gives a sense for the variance involved in our approach. However, more runs would be preferred.

We have placed emphasis on evaluating our approach on samples that have not been seen previously. This ensures that our agents have not simply remembered the training levels, but actually generalized to unseen samples. One issue we would like to note is that we have used a relatively small number of test samples when measuring search performance. This is to make it feasible to collect results for human searchers. Ideally, we would have tested on a larger set.

The three metrics we have used (success rate, average search path length, and SPL) all offer different insights into the performance of agents. They should not be used as absolute measures, but rather to compare different agents in the same environments. The SPL metric has been used in several previous works for navigation tasks [4, 66]. SPL considers both search path length, shortest path length and whether the agent was successful in each episode, and is useful as a measure for the quality of the search paths chosen by an agent.

However, SPL is not a perfect metric. As noted by [11], it fails to consider the fact that some failures are less of a failure than others. For example, an agent might have been close to discovering a target at some time step but missed it. It might be desirable to give a higher score to such search paths than ones that are never close to targets. The binary success indicator introduces high variance into the metric. They also note that it is not suitable for comparison across different datasets, as obtaining high SPL scores is more difficult for short paths. All of our agents have achieved relatively high success rates, so the issue of failure does not seem like a major concern. We have also only compared SPL within the same datasets.

Finally, there is a possibility that there are errors in our implementation. We have tested our PPO implementation against standard tasks, but there is still the possibility of errors in the algorithm, environments or evaluation. We have provided source code for our implementation and invite others to reproduce or scrutinize our results.

5.2.6 Source Criticism

- Some cited papers are preprints.
- Several of the cited preprints have actually been published in journals, replace these.

5.3 The work in a wider context

A major concern of artificial intelligence researchers and the broader public today is the implications of autonomous systems…

- [54], [62], [1], [53], [15]
- `https://people.eecs.berkeley.edu/~russell/research/LAWS.html`

- Dangers of lethal autonomous weapon systems
- Search-and-rescue and fire detection require high reliability, and a short search time can be the difference between life and death.
- Things to consider before deploying a system for such a scenario include verification, validity, security, control, human intervention, etc.

"As the authors of this work, we explicitly condemn its use for lethal weaponry."

6 Conclusion

Visual search is ubiquitous in our daily lives as humans. Automated visual search systems therefore naturally have many potential applications. …

In this work, we have presented a method for jointly learning control of visual attention, recognition and localization using deep reinforcement learning. …

- Relate back to research questions.
- Answer them explicitly.

6.1 Future Work

6.1.1 Realistic Search Tasks

- Real search problems are likely to be more difficult to learn.
- There is a lack of datasets and environments for this (AI2 THOR, object detection in large images)

6.1.2 Ablation Studies

- Evaluate the importance of image observations, position observations and memory.
- We could draw some conclusions regarding these from our results…

A Learning Curves

To do...

B Search Paths

To do...

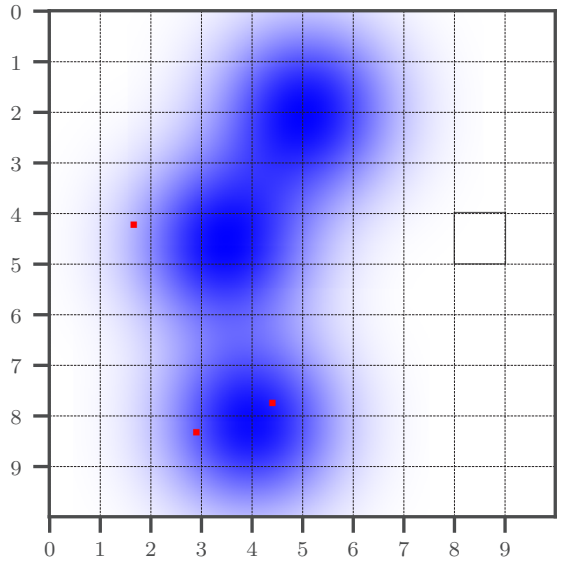

Figure B.1: Scene from gaussian environment used for search path sample visualizations.

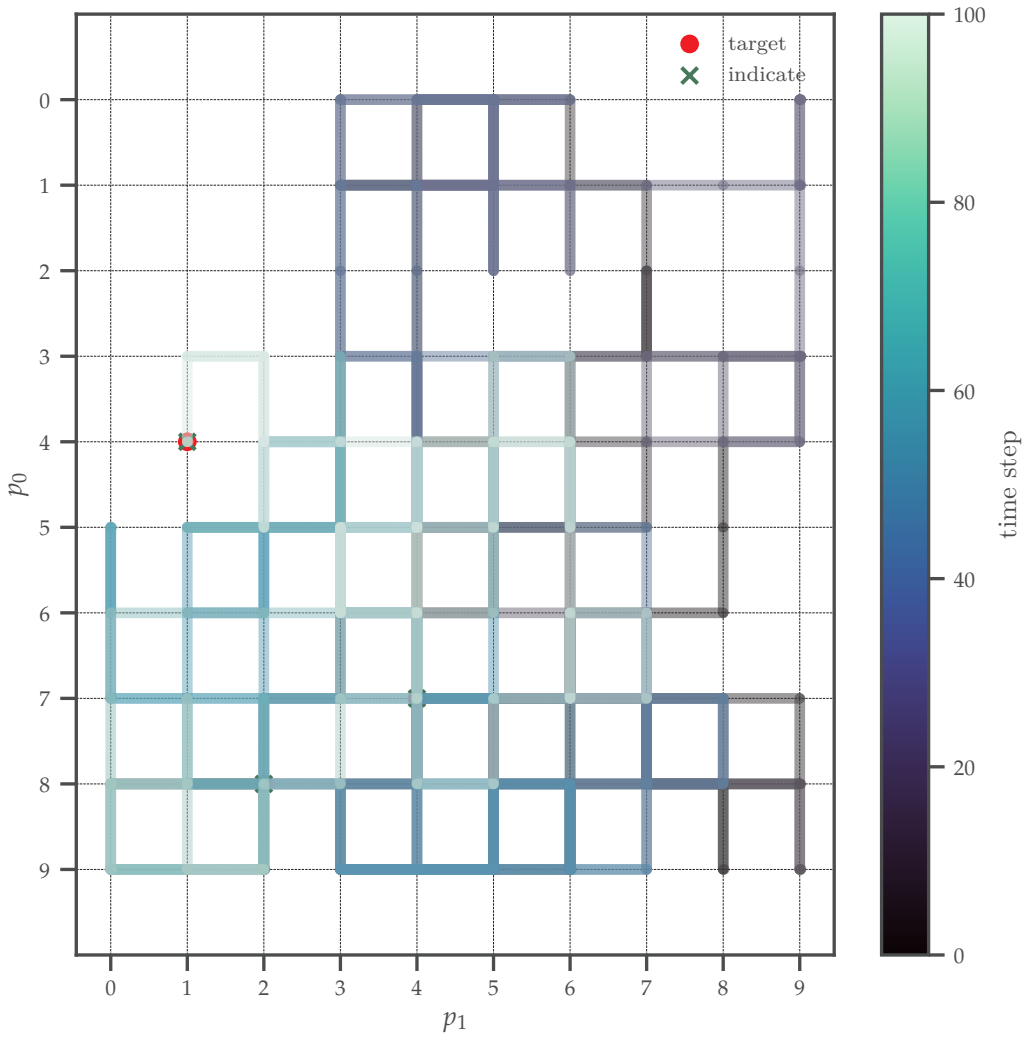

Figure B.2: Sample search path for random baseline policy in gaussian environment.

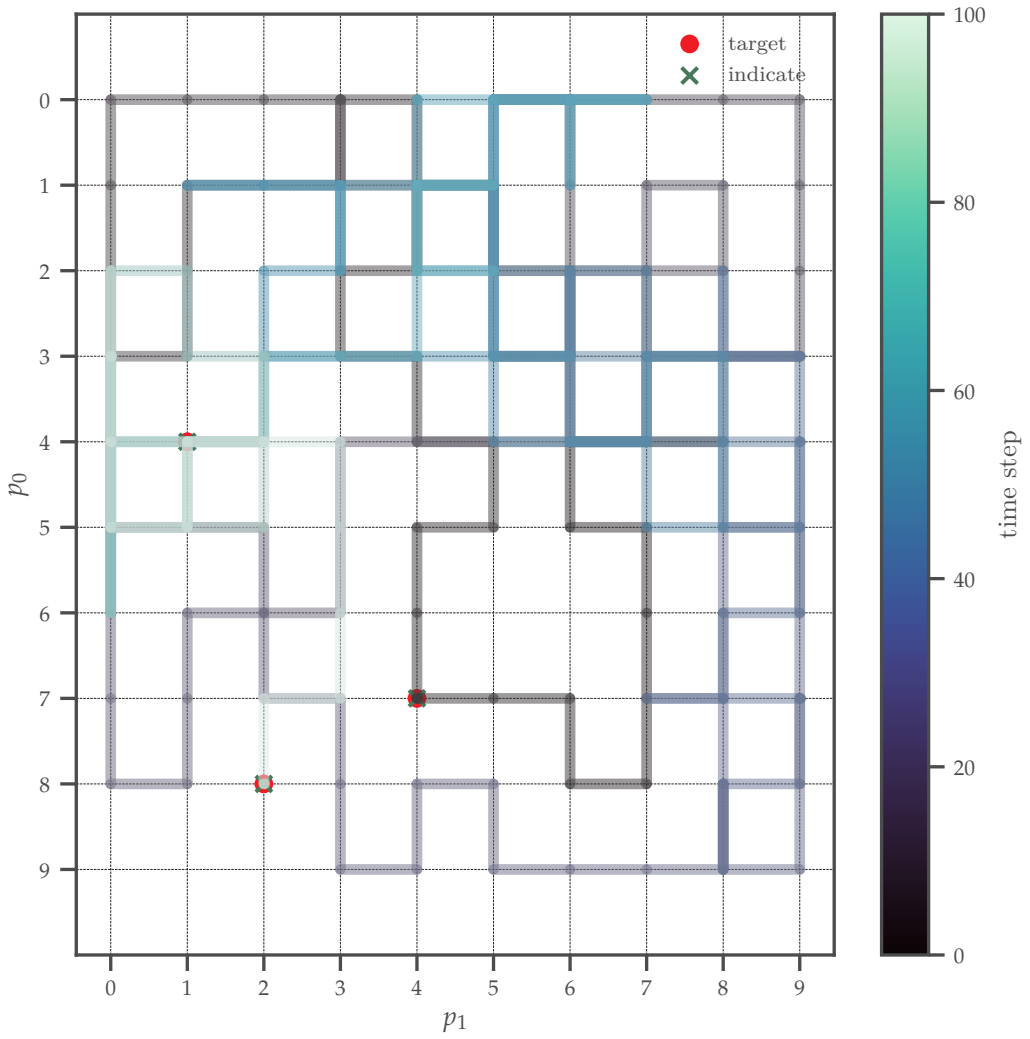

Figure B.3: Sample search path for greedy baseline policy in gaussian environment.

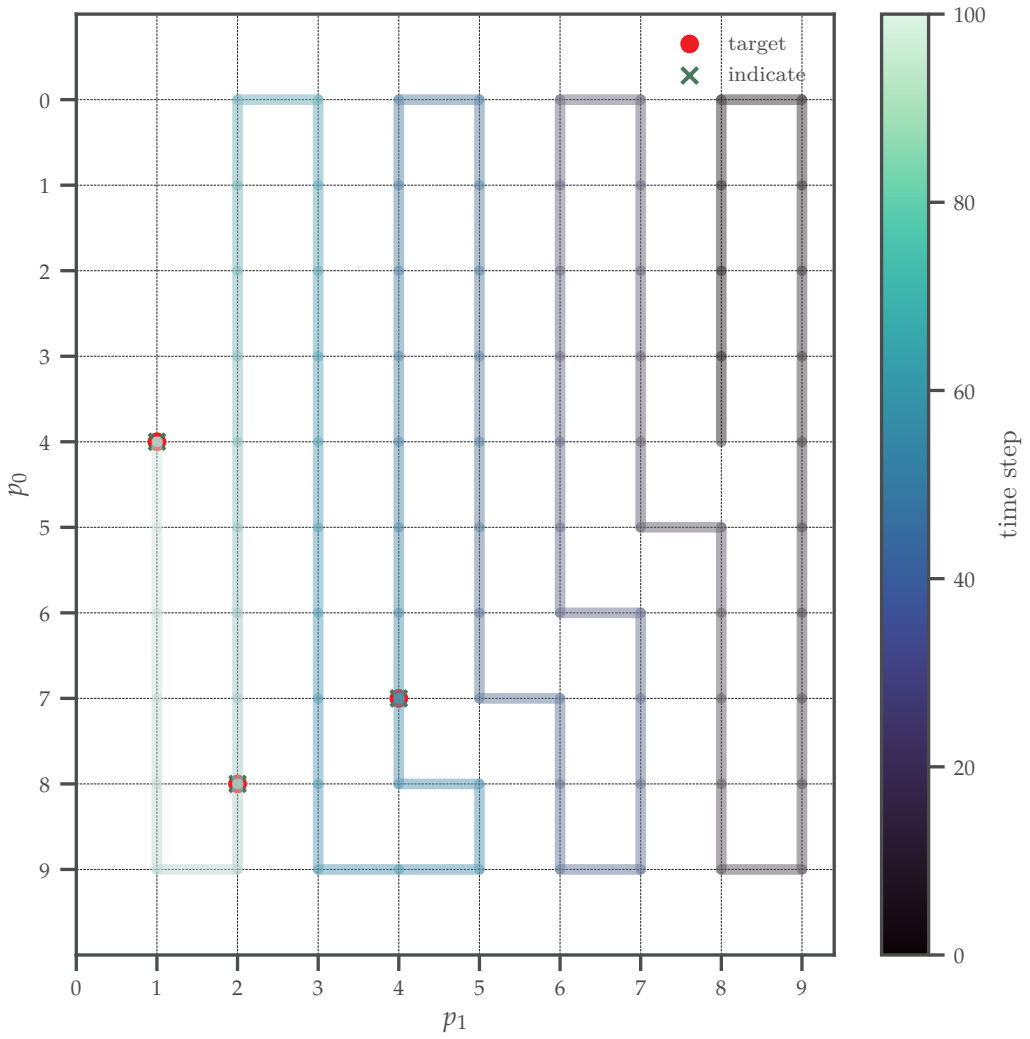

Figure B.4: Sample search path for exhaustive baseline policy in gaussian environment.

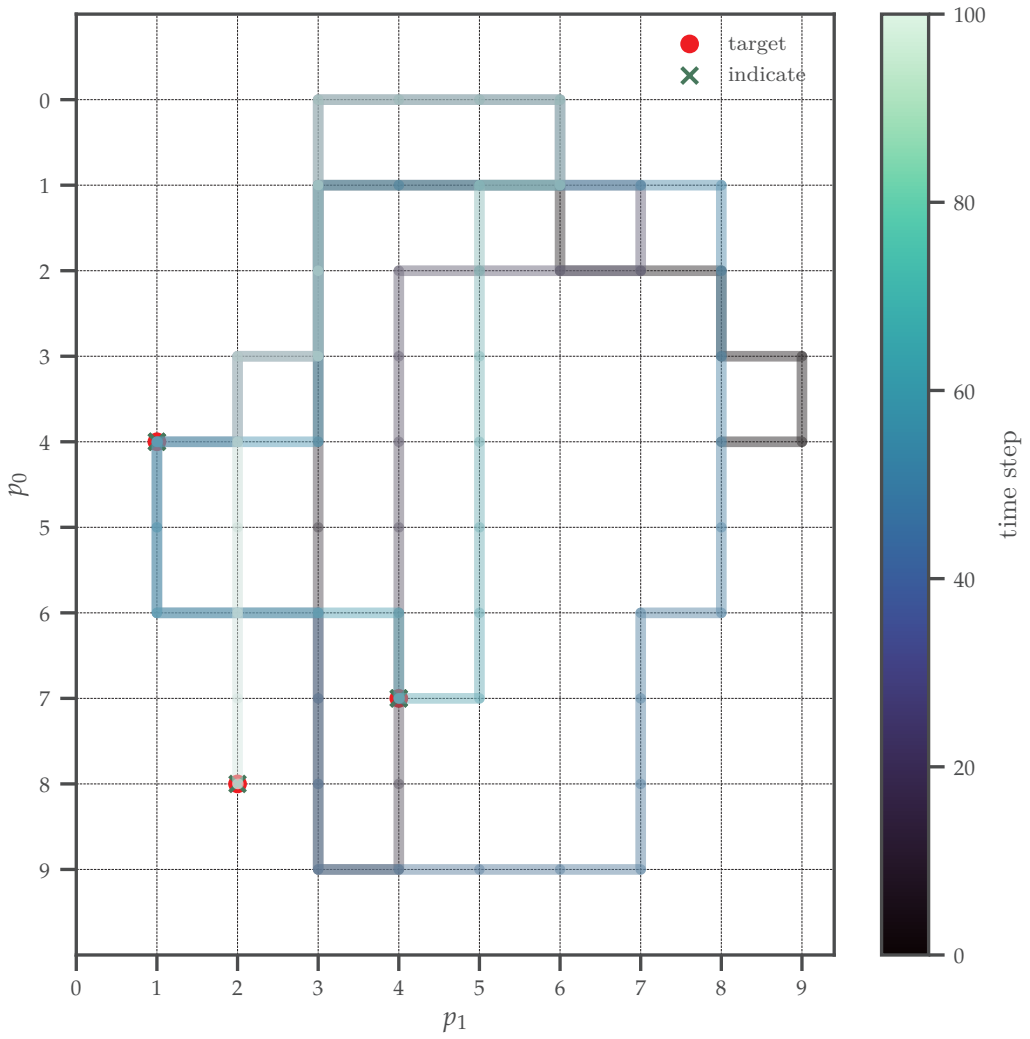

Figure B.5: Sample search path for temporal memory agent in gaussian environment.

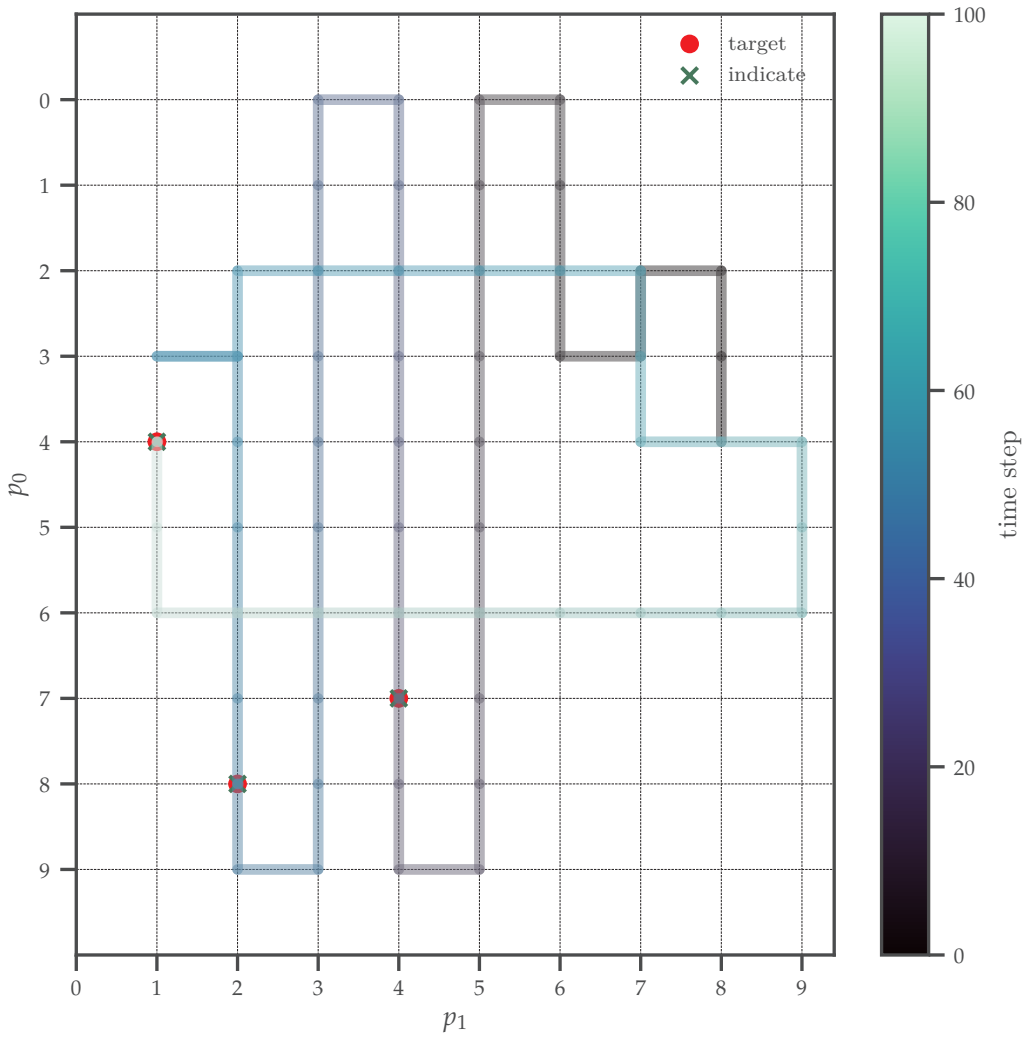

Figure B.6: Sample search path for spatial memory agent in gaussian environment.

Bibliography

[1] en. In: *Nature* 521.75537553 (May 2015), pp. 415–418. ISSN: 1476-4687. DOI: 10.1038/521415a (Cited on page 35).

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *arXiv:2108.13264 [cs, stat]* (Jan. 2022). arXiv: 2108.13264. URL: http://arxiv.org/abs/2108.13264 (Cited on pages 16, 23, 35).

[3] John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. "Active vision". In: *International Journal of Computer Vision* 1.4 (Jan. 1988). 705 citations (Crossref) [2022-02-07], pp. 333–356. ISSN: 0920-5691, 1573-1405. DOI: 10/cn4mdc. URL: http://link.springer.com/10.1007/BF00133571 (visited on 02/07/2022) (Cited on page 5).

[4] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir. "On Evaluation of Embodied Navigation Agents". In: *arXiv:1807.06757 [cs]* (July 2018). arXiv: 1807.06757. URL: http://arxiv.org/abs/1807.06757 (Cited on pages 13, 16, 25, 35).

[5] Alexander Andreopoulos and John K Tsotsos. "A Theory of Active Object Localization". en. In: (), p. 8 (Cited on page 5).

[6] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphael Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. "What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study". In: *arXiv:2006.05990 [cs, stat]* (June 2020). arXiv: 2006.05990. URL: http://arxiv.org/abs/2006.05990 (Cited on pages 10, 23).

[7] A. Aydemir, K. Sjoo, J. Folkesson, A. Pronobis, and P. Jensfelt. "Search in the real world: Active visual object search based on spatial relations". en. In: *2011 IEEE International Conference on Robotics and Automation*. 48 citations (Crossref) [2022-02-28]. Shanghai, China: IEEE, May 2011, pp. 2818–2824. ISBN: 978-1-61284-386-5. DOI: 10.1109/ICRA.2011.5980495. URL: http://ieeexplore.ieee.org/document/5980495/ (Cited on page 11).

[8] Alper Aydemir, Andrzej Pronobis, Moritz Göbelbecker, and Patric Jensfelt. "Active Visual Object Search in Unknown Environments Using Uncertain Semantics". In: *IEEE Transactions on Robotics* 29.4 (Aug. 2013). 65 citations (Crossref) [2022-02-28], pp. 986–1002. ISSN: 1941-0468. DOI: 10.1109/TRO.2013.2256686 (Cited on page 11).

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv:1409.0473 [cs, stat]* (May 2016). arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473 (Cited on page 14).

[10] R. Bajcsy. "Active perception". In: *Proceedings of the IEEE* 76.8 (Aug. 1988). 646 citations (Crossref) [2022-02-28] Conference Name: Proceedings of the IEEE, pp. 966–1005. ISSN: 1558-2256. DOI: 10.1109/5.5968 (Cited on page 5).

[11] Dhruv Batra, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans. "ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects". In: *arXiv:2006.13171 [cs]* (Aug. 2020). arXiv: 2006.13171. URL: http://arxiv.org/abs/2006.13171 (Cited on page 35).

[12] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *arXiv:1206.5538 [cs]* (Apr. 2014). arXiv: 1206.5538. URL: http://arxiv.org/abs/1206.5538 (Cited on page 5).

[13] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. "Visual Navigation for Mobile Robots: A Survey". In: *Journal of Intelligent and Robotic Systems* 53.3 (Nov. 2008), pp. 263–296. ISSN: 0921-0296, 1573-0409. DOI: 10.1007/s10846-008-9235-4 (Cited on page 12).

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. "OpenAI Gym". In: *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540. URL: http://arxiv.org/abs/1606.01540 (Cited on page 25).

[15] Miles Brundage, Shahar Avin, Jack Clark, Helen Toner, Peter Eckersley, Ben Garfinkel, Allan Dafoe, Paul Scharre, Thomas Zeitzoff, Bobby Filar, Hyrum Anderson, Heather Roff, Gregory C. Allen, Jacob Steinhardt, Carrick Flynn, Seán Ó hÉigeartaigh, Simon Beard, Haydn Belfield, Sebastian Farquhar, Clare Lyle, Rebecca Crootof, Owain Evans, Michael Page, Joanna Bryson, Roman Yampolskiy, and Dario Amodei. *The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation*. arXiv:1802.07228. arXiv:1802.07228 [cs] type: article. Feb. 2018. DOI: 10.48550/arXiv.1802.07228. URL: http://arxiv.org/abs/1802.07228 (Cited on page 35).

[16] Juan C. Caicedo and Svetlana Lazebnik. "Active Object Localization with Deep Reinforcement Learning". In: *arXiv:1511.06015 [cs]* (Nov. 18, 2015). arXiv: 1511.06015. URL: http://arxiv.org/abs/1511.06015 (visited on 02/03/2022) (Cited on pages 2, 11–13).

[17] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov. "Object Goal Navigation using Goal-Oriented Semantic Exploration". In: *arXiv:2007.00643 [cs]* (July 2020). arXiv: 2007.00643. URL: http://arxiv.org/abs/2007.00643 (Cited on pages 14, 22).

[18] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. "Active vision in robotic systems: A survey of recent developments". In: *The International Journal of Robotics Research* 30.11 (Sept. 2011), pp. 1343–1377. ISSN: 0278-3649. DOI: 10.1177/0278364911410755 (Cited on pages 5, 11).

[19] Xinlei Chen and Abhinav Gupta. "Spatial Memory for Context Reasoning in Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, Oct. 2017, pp. 4106–4116. ISBN: 978-1-5386-1032-9. DOI: 10.1109/ICCV.2017.440. URL: http://ieeexplore.ieee.org/document/8237702/ (Cited on page 12).

[20] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. "Leveraging Procedural Generation to Benchmark Reinforcement Learning". In: *arXiv:1912.01588 [cs, stat]* (July 2020). arXiv: 1912.01588. URL: http://arxiv.org/abs/1912.01588 (Cited on pages 10, 15, 17, 23, 25, 33).

[21] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. "Quantifying Generalization in Reinforcement Learning". In: *arXiv:1812.02341 [cs, stat]* (July 2019). arXiv: 1812.02341. URL: http://arxiv.org/abs/1812.02341 (Cited on pages 15, 33).

[22] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. "A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms". en. In: *arXiv:1904.06979 [cs, stat]* (Apr. 2019). arXiv: 1904.06979. URL: http://arxiv.org/abs/1904.06979 (Cited on pages 16, 23, 35).

[23] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. "How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments". en. In: *arXiv:1806.08295 [cs, stat]* (July 2018). arXiv: 1806.08295. URL: http://arxiv.org/abs/1806.08295 (Cited on page 16).

[24] Vikas Dhiman, Shurjo Banerjee, Brent Griffin, Jeffrey M. Siskind, and Jason J. Corso. "A Critical Investigation of Deep Reinforcement Learning for Navigation". In: *arXiv:1802.02274 [cs]* (Jan. 2019). arXiv: 1802.02274. URL: http://arxiv.org/abs/1802.02274 (Cited on pages 13, 34).

[25] M. P. Eckstein. "Visual search: A retrospective". In: *Journal of Vision* 11.5 (Dec. 30, 2011). 207 citations (Crossref) [2022-02-28], pp. 14–14. ISSN: 1534-7362. DOI: 10.1167/11.5.14. URL: http://jov.arvojournals.org/Article.aspx?doi=10.1167/11.5.14 (visited on 02/22/2022) (Cited on pages 1, 5).

[26] Per-Erik Forssen, David Meger, Kevin Lai, Scott Helmer, James J. Little, and David G. Lowe. "Informed visual search: Combining attention and object recognition". In: *2008 IEEE International Conference on Robotics and Automation*. May 2008, pp. 935–942. DOI: 10.1109/ROBOT.2008.4543325 (Cited on pages 2, 10).

[27] Enric Galceran and Marc Carreras. "A survey on coverage path planning for robotics". In: *Robotics and Autonomous Systems* 61.12 (Dec. 2013), pp. 1258–1276. ISSN: 09218890. DOI: 10/f5j2n5 (Cited on page 5).

[28] Florin C. Ghesu, Bogdan Georgescu, Tommaso Mansi, Dominik Neumann, Joachim Hornegger, and Dorin Comaniciu. "An Artificial Agent for Anatomical Landmark Detection in Medical Images". In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016*. Ed. by Sebastien Ourselin, Leo Joskowicz, Mert R. Sabuncu, Gozde Unal, and William Wells. Vol. 9902. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 229–237. ISBN: 978-3-319-46725-2. DOI: 10.1007/978-3-319-46726-9_27. URL: https://link.springer.com/10.1007/978-3-319-46726-9_27 (Cited on pages 2, 12, 13).

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, Nov. 2016. ISBN: 978-0-262-03561-3 (Cited on pages 2, 5–7).

[30] Saurabh Gupta, Varun Tolani, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. "Cognitive Mapping and Planning for Visual Navigation". In: *arXiv:1702.03920 [cs]* (Feb. 2019). arXiv: 1702.03920. URL: http://arxiv.org/abs/1702.03920 (Cited on pages 14, 22).

[31] Matthew Hausknecht and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *arXiv:1507.06527 [cs]* (Jan. 2017). arXiv: 1507.06527. URL: http://arxiv.org/abs/1507.06527 (Cited on pages 13, 14, 22).

[32] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep Reinforcement Learning That Matters". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.11 (Apr. 2018). ISSN: 2374-3468. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11694 (Cited on pages 10, 16, 22, 23, 35).

[33] Joao F. Henriques and Andrea Vedaldi. "MapNet: An Allocentric Spatial Memory for Mapping Environments". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, June 2018, pp. 8476–8484. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00884. URL: https://ieeexplore.ieee.org/document/8578982/ (Cited on pages 14, 22).

[34] Matteo Hessel, Hado van Hasselt, Joseph Modayil, and David Silver. "On Inductive Biases in Deep Reinforcement Learning". In: *arXiv:1907.02908 [cs, stat]* (July 2019). arXiv: 1907.02908. URL: http://arxiv.org/abs/1907.02908 (Cited on pages 15, 34).

[35] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735 (Cited on pages 7, 22).

[36] Laurent Itti and Christof Koch. "Computational modelling of visual attention". In: *Nature Reviews Neuroscience* 2.33 (Mar. 2001), pp. 194–203. ISSN: 1471-0048. DOI: 10.1038/35058500 (Cited on pages 4, 5).

[37] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1–2 (May 1998), pp. 99–134. ISSN: 00043702. DOI: 10.1016/S0004-3702(98)00023-X (Cited on pages 7, 8).

[38] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (Cited on page 23).

[39] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. "A Survey of Generalisation in Deep Reinforcement Learning". In: *arXiv:2111.09794 [cs]* (Jan. 2022). arXiv: 2111.09794. URL: http://arxiv.org/abs/2111.09794 (Cited on pages 14, 15, 17).

[40] Maja J Mataric. "Reward Functions for Accelerated Learning". In: *Machine Learning Proceedings 1994*. Ed. by William W. Cohen and Haym Hirsh. Morgan Kaufmann, Jan. 1994, pp. 181–189. ISBN: 978-1-55860-335-6. DOI: 10.1016/B978-1-55860-335-6.50030-1. URL: https://www.sciencedirect.com/science/article/pii/B9781558603356500301 (Cited on page 9).

[41] Marvin Minsky. "Steps toward Artificial Intelligence". In: *Proceedings of the IRE* 49.1 (Jan. 1961), pp. 8–30. ISSN: 2162-6634. DOI: 10.1109/JRPROC.1961.287775 (Cited on page 9).

[42] Silviu Minut and Sridhar Mahadevan. "A reinforcement learning model of selective visual attention". In: *Proceedings of the fifth international conference on Autonomous agents - AGENTS '01*. ACM Press, 2001, pp. 457–464. ISBN: 978-1-58113-326-4. DOI: 10/dbwckq. URL: http://portal.acm.org/citation.cfm?doid=375735.376414 (Cited on pages 2, 11).

[43] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. "Learning to Navigate in Complex Environments". In: *arXiv:1611.03673 [cs]* (Jan. 2017). arXiv: 1611.03673. URL: http://arxiv.org/abs/1611.03673 (Cited on pages 2, 13, 22).

[44] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous Methods for Deep Reinforcement Learning". In: *arXiv:1602.01783 [cs]* (June 2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783 (Cited on pages 13, 14, 17, 22).

[45] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. "Recurrent Models of Visual Attention". In: *arXiv:1406.6247 [cs, stat]* (June 2014). arXiv: 1406.6247. URL: http://arxiv.org/abs/1406.6247 (Cited on pages 2, 11, 14).

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning". In: *arXiv:1312.5602 [cs]* (Dec. 2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602 (Cited on pages 9, 23).

[47] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. "Human-level control through deep reinforcement learning". In: *Nature* 518.75407540 (Feb. 2015), pp. 529–533. ISSN: 1476-4687. DOI: 10.1038/nature14236 (Cited on pages 2, 9, 13, 14, 21).

[48] Ken Nakayama and Paolo Martini. "Situating visual search". In: *Vision Research*. Vision Research 50th Anniversary Issue: Part 2 51.13 (July 2011), pp. 1526–1537. ISSN: 0042-6989. DOI: 10.1016/j.visres.2010.09.003 (Cited on pages 1, 4).

[49] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. "Control of Memory, Active Perception, and Action in Minecraft". In: *arXiv:1605.09128 [cs]* (May 2016). arXiv: 1605.09128. URL: http://arxiv.org/abs/1605.09128 (Cited on pages 14, 32).

[50] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. "Time Limits in Reinforcement Learning". In: *arXiv:1712.00378 [cs]* (Jan. 2022). arXiv: 1712.00378. URL: http://arxiv.org/abs/1712.00378 (Cited on page 18).

[51] Emilio Parisotto and Ruslan Salakhutdinov. "Neural Map: Structured Memory for Deep Reinforcement Learning". In: *arXiv:1702.08360 [cs]* (Feb. 2017). arXiv: 1702.08360. URL: http://arxiv.org/abs/1702.08360 (Cited on pages 14, 22, 33, 34).

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: (), p. 12 (Cited on page 25).

[53] Stuart Russell. "Provably Beneficial Artificial Intelligence". In: *27th International Conference on Intelligent User Interfaces*. IUI '22. New York, NY, USA: Association for Computing Machinery, Mar. 2022, p. 3. ISBN: 978-1-4503-9144-3. DOI: 10.1145/3490099.3519388. URL: https://doi.org/10.1145/3490099.3519388 (Cited on page 35).

[54] Stuart Russell, Daniel Dewey, and Max Tegmark. "Research Priorities for Robust and Beneficial Artificial Intelligence". en. In: *AI Magazine* 36.44 (Dec. 2015), pp. 105–114. ISSN: 2371-9621. DOI: 10.1609/aimag.v36i4.2577 (Cited on page 35).

[55] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach*. In collab. with Ming-wei Chang, Jacob Devlin, Anca Dragan, David Forsyth, Ian Goodfellow, Jitendra Malik, Vikash Mansinghka, Judea Pearl, and Michael Woolridge. Fourth Edition. Pearson Series in Artificial Intelligence. Hoboken, NJ: Pearson, 2021. 1115 pp. ISBN: 978-0-13-461099-3 (Cited on pages 5–7).

[56] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal Policy Optimization Algorithms". In: *arXiv:1707.06347 [cs]* (Aug. 2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347 (Cited on pages 10, 23, 25).

[57] Ksenia Shubina and John K. Tsotsos. "Visual search for an object in a 3D environment using a mobile robot". In: *Computer Vision and Image Understanding*. Special issue on Intelligent Vision Systems 114.5 (May 2010), pp. 535–547. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2009.06.010 (Cited on pages 2, 11).

[58] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.75877587 (Jan. 2016), pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961 (Cited on pages 2, 35).

[59] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 1999. URL: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf (Cited on page 8).

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Second edition. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018. 526 pp. ISBN: 978-0-262-03924-6 (Cited on pages 2, 7–9).

[61] Gerald Tesauro et al. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68 (Cited on page 9).

[62] Ricardo Vinuesa, Hossein Azizpour, Iolanda Leite, Madeline Balaam, Virginia Dignum, Sami Domisch, Anna Felländer, Simone Daniela Langhans, Max Tegmark, and Francesco Fuso Nerini. "The role of artificial intelligence in achieving the Sustainable Development Goals". en. In: *Nature Communications* 11.11 (Jan. 2020), p. 233. ISSN: 2041-1723. DOI: 10.1038/s41467-019-14108-y (Cited on page 35).

[63] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.77827782 (Nov. 2019), pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z (Cited on pages 2, 10, 35).

[64] Jeremy M. Wolfe. "Visual search". In: *Current biology : CB* 20.8 (Apr. 27, 2010). 64 citations (Crossref) [2022-03-02] Publisher: NIH Public Access, R346. DOI: 10.1016/j.cub.2010.02.016. URL: https://www.ncbi.nlm.nih.gov/labs/pmc/articles/PMC5678963/ (visited on 03/02/2022) (Cited on pages 1, 4).

[65] Jeremy M. Wolfe and Todd S. Horowitz. "Five factors that guide attention in visual search". In: *Nature Human Behaviour* 1.3 (Mar. 8, 2017). 300 citations (Crossref) [2022-02-28] Number: 3 Publisher: Nature Publishing Group, pp. 1–8. ISSN: 2397-3374. DOI: 10.1038/s41562-017-0058. URL: https://www.nature.com/articles/s41562-017-0058 (visited on 02/28/2022) (Cited on pages 1, 5).

[66] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta, and Roozbeh Mottaghi. "Visual Semantic Navigation using Scene Priors". In: *arXiv:1810.06543 [cs]* (Oct. 2018). arXiv: 1810.06543. URL: http://arxiv.org/abs/1810.06543 (Cited on pages 13, 35).

[67] Xin Ye, Zhe Lin, Haoxiang Li, Shibin Zheng, and Yezhou Yang. "Active Object Perceiver: Recognition-Guided Policy Learning for Object Searching on Mobile Robots". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Oct. 2018, pp. 6857–6863. DOI: 10.1109/IROS.2018.8593720 (Cited on page 12).

[68] Yiming Ye and John K. Tsotsos. "A Complexity-Level Analysis of the Sensor Planning Task for Object Search". en. In: *Computational Intelligence* 17.4 (Nov. 2001). 13 citations (Crossref) [2022-02-28], pp. 605–620. ISSN: 0824-7935, 1467-8640. DOI: 10.1111/0824-7935.00166 (Cited on pages 5, 11).

[69] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. "A Study on Overfitting in Deep Reinforcement Learning". In: *arXiv:1804.06893 [cs, stat]* (Apr. 2018). arXiv: 1804.06893. URL: http://arxiv.org/abs/1804.06893 (Cited on pages 14, 33).

[70] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. "Object Detection With Deep Learning: A Review". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.11 (Nov. 2019), pp. 3212–3232. ISSN: 2162-237X, 2162-2388. DOI: 10.1109/TNNLS.2018.2876865 (Cited on page 11).

[71] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. "Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning". In: *arXiv:1609.05143 [cs]* (Sept. 16, 2016). arXiv: 1609.05143. URL: http://arxiv.org/abs/1609.05143 (visited on 03/14/2022) (Cited on pages 2, 12).