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

# Learning to Search for Targets

– A Deep Reinforcement Learning Approach for 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

## Upphovsrätt



## Copyright

**Abstract**

The abstract resides in file `Abstract.tex`. Here you should write a short summary of your work.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in massa suscipit, congue massa in, pharetra lacus. Donec nec felis tempor, suscipit metus molestie, consectetur orci. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Curabitur fermentum, augue non ullamcorper tempus, ex urna suscipit lorem, eu consectetur ligula orci quis ex. Phasellus imperdiet dolor at luctus tempor. Curabitur nisi enim, porta ut gravida nec, feugiat fermentum purus. Donec hendrerit justo metus. In ultrices malesuada erat id scelerisque. Sed sapien nisi, feugiat in ligula vitae, condimentum accumsan nisi. Nunc sit amet est leo. Quisque hendrerit, libero ut viverra aliquet, neque mi vestibulum mauris, a tincidunt nulla lacus vitae nunc. Cras eros ex, tincidunt ac porta et, vulputate ut lectus. Curabitur ultricies faucibus turpis, ac placerat sem sollicitudin at. Ut libero odio, eleifend in urna non, varius imperdiet diam. Aenean lacinia dapibus mauris. Sed posuere imperdiet ipsum a fermentum.

Nulla lobortis enim ac magna rhoncus, nec condimentum erat aliquam. Nullam laoreet interdum lacus, ac rutrum eros dictum vel. Cras lobortis egestas lectus, id varius turpis rhoncus et. Nam vitae auctor ligula, et fermentum turpis. Morbi neque tellus, dignissim a cursus sed, tempus eu sapien. Morbi volutpat convallis mauris, a euismod dui egestas sit amet. Nullam a volutpat mauris. Fusce sed ipsum lectus. In feugiat, velit eu fermentum efficitur, mi ex eleifend ante, eget scelerisque sem turpis nec augue.

Vestibulum posuere nibh ut iaculis semper. Ut diam justo, interdum quis felis ac, posuere fermentum ex. Fusce tincidunt vel nunc non semper. Sed ultrices suscipit dui, vel lacinia lorem euismod quis. Etiam pellentesque vitae sem eu bibendum. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Pellentesque scelerisque congue ullamcorper. Sed vehicula sodales velit a scelerisque. Pellentesque dignissim lectus ipsum, quis consectetur tellus rhoncus a.

Nunc placerat ut lectus vel ornare. Sed nec dictum enim. Donec imperdiet, ipsum ut facilisis blandit, lacus nisi maximus ex, sed semper nisl metus eget leo. Nunc efficitur risus ac risus placerat, vel ullamcorper felis interdum. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Duis vitae felis vel nibh sodales fringilla. Donec semper eleifend sem quis ornare. Proin et leo ut dolor consectetur vehicula. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Nunc dignissim interdum orci, sit amet pretium nibh consectetur sagittis. Aenean a eros id risus aliquam placerat nec ut lectus. Curabitur at quam in nisi sodales imperdiet in at erat. Praesent euismod pulvinar imperdiet. Nam auctor mattis nisi in efficitur. Quisque non cursus ipsum, consequat vehicula justo. Fusce varius metus et nulla rutrum scelerisque. Praesent molestie elementum nulla a consequat. In at facilisis nisi, convallis molestie sapien. Cras id ullamcorper purus. Sed at lectus sit amet dolor finibus suscipit vel et purus. Sed odio ipsum, dictum vel justo sit amet, interdum dictum justo. Quisque euismod quam magna, at dignissim eros varius in. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

# Acknowledgments

Acknowledgments.tex

# Contents

# List of Figures

# List of Tables

# Todo list

Describe approach once it is fully decided. Will be a memory. . . . . 20

# Notation

| | |
|---|---|
| $x$ | variable |
| $X$ | random variable |
| $\vec{x}$ | vector |
| $\mathbb{X}$ | set |

# 1 Introduction

In this thesis project, the problem of searching for targets 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 behaviour. 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.

An intelligent searcher should be able to

Automating the task of searching is of great interest

While searching for targets is often seemingly effortless to humans, it is a complex process. How humans and animals search for things has been extensively studied in neuroscience and neurobiology [17, 48, 47].

Applications such as helping robots and search and rescue mean that it is of great interest to automate visual search. In the computer vision field, there has been several attempts to mimic the way humans search in machines []. Most attempts focus on fully observable scenes where the target is in view and the task is to localize it (object localization). However, in many real-world visual search scenarios the field-of-view is limited. This means that the search process is split into two steps: directing the field of view (covert attention), and locating targets within the view (overt attention). Much work has been focused on latter, locating targets within the field of view [].

When only a fraction of the environment is visible, where to move the field of view becomes an important decision. The characteristics of the searched environment can often be used to find targets quicker. For example, if one is foraging for blueberries it makes sense to search the ground rather than the trees. Similarly, if one is searching a satellite image for boats it is reasonable to focus on ocean shores. If you see a railroad track or the wake of a boat you can usually follow it to find a vehicle. The exact characteristics of the environment need not be constant - forests with blueberries can vary greatly in appearance and boats can be found in all of the seven seas. In many cases, the environment is familiar in that it has char-

acteristics that are similar to previously seen environments. Humans are able to generalize in such cases.

Manually creating search algorithms for such tasks is problematic. The appearance and distribution of targets in an environment varies greatly, and may be subtle. The visual richness of the environment itself is another problem. How can you identify useful hints from the environment to guide covert attention? Doing so manually can be labour intensive, especially if a searcing system should be deployed in many different environments. If one could instead learn the underlying from a limited set of sample environments and generalize to unseen similar environments this problem would be circumvented.

Deep reinforcement learning is an approach for how to act. It has been applied to a number of problems with sucess...

## 1.2 Aim

The aim of this thesis is to investigate how an intelligent agent that learns to search for targets 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 effectively search for targets in unseen environments. Specifically, we consider scenarious where the agent can only observe a small portion of its environment at any given time. The agent has to actively choose where to look in order to gain new information about the environment.

We postulate that an effective searcher

- prioritizes regions where the probability of finding a target is high according to previous experience,
- is able to search the environment exhaustively,
- avoids searching the same area twice unless,
- learns how the distribution of targets is correlated to the appearance of the environment,
- utilizes information from previously visited regions to decide where to look, and
- is able to find multiple targets while minimizing its path length.

Our contributions are as follows:

- We provide a set of environments to evaluate visual search agents.
- We propose a method for solving the visual search task with reinforcement learning.
- We compare the method to a set of common baseline agents.

## 1.3 Research questions

This thesis will address the following questions:

1. How can a learning agent that learns to intelligently search for targets be implemented?
2. How does the learning agent compare to random walk, exhaustive search, a human searcher?
3. How well does the learning agent generalize to unseen but familiar environments?

## 1.4 Delimitations

This thesis will be focused on the behavioral aspects of the presented problem. We do not focus on difficult detection problems, but rather efficient actions. For this reason, targets will deliberately be made easy to detect. For simplicity, 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.

# 2 Theory

This chapter introduces background and related work.

## 2.1 Background

### 2.1.1 Active Vision

Much of past and present research in machine perception involves a passive observer. Images are passively sampled and perceived. Animal perception, however, is active. We do not only see things, but look for them. One might ask why this is the case, if there is any advantage that an active observer has over a passive one. Aloimonos and Weiss (1988) [2] introduce the paradigm called *active vision*, and prove that an active observer can solve several basic vision problems in a more efficient way than a passive one.

Bajcsy (1988) [**bajcsy_1988**] defines active vision, and perception in general, as a problem of intelligent data acquisition. An active observer needs to define and measure parameters and errors from its scene and feed then back to control the data acquisition process. Bajscy states that one of the difficulties of this problem 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. One view lacks information that may be present with multiple views. Multiple views also add the time dimension into the problem.

In a re-visitation of active perception, Bajcsy, Aloimonos and Tsotsos (2018) [**bajcsy_aloimonos_tsotsos_2018**] stress that despite recent successes in robotics, artificial intelligence and computer vision, an intelligent agent must include active perception:

> An agent is an active perceiver if it knows why it wishes to sense, and then chooses what to perceive, and determines how, when and where to achieve that perception

[**bajcsy_aloimonos_tsotsos_2018**]
[12]

### 2.1.2 Visual Search

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

Wolfe (2021) [47] describes a model of visual search

Eckstein (2011) [17] reviews efforts from various subfields and identifies a set of mechanisms used to achieve efficient visual search. Knowledge about the target, distractor, background statistical properties, location probabilities, contextual cues, rewards and target prevalence are all identified as useful. This is motivated with evidence from psychology as well as neural correlates.

Visual search is not always instant, and can in fact often be slow. This is in part due to processing: our visual system cannot process the entire visual field and

Wolfe and Horowitz (2017) [**wolfe_horowitz_2017**] identify and measure a set of factors that guide attention in visual search. One of these is bottom-up guidance, in which some visual properties of the scene draw more attention than others. Another is top-down guidance, which is user driven and directed to objects with known features of desired targets. Scene guidance is also identified, in which attributes of the scene guide attention to areas likely to contain targets.

These works ground the task considered in this project in psychology.

### 2.1.3 Visual Attention

...

### 2.1.4 Reinforcement Learning

Reinforcement learning (RL) [45] is a subfield of machine learning concerned with learning from interaction how to achieve a goal. This section introduces the fundamental concepts of RL.

#### 2.1.4.1 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. [28]

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,
- $\Omega$ is a finite set of observations,

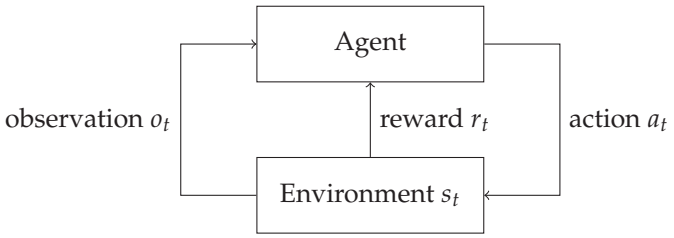


Figure 2.1: Partially observable Markov decision process.

- $\mathcal{O} : \mathcal{S} \times \mathcal{A} \to \Pi(\Omega)$ 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 $\mathcal{O}(s_{t+1}, a_t, o_t)$. [28] Figure 2.1.4.1 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*:

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

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 them is often computationally intractable. Approximate solutions are more common, where the agent usually maintains an internal *belief state* [28] 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 $s$ interchangably.

#### 2.1.4.2 Policies and Value Functions

The behaviour of the agent is described by its *policy*. A policy $\pi$ 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 $s$. [45]

Most RL solutions methods also approximate a *value function*. A value function $v_\pi$ 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 $\pi$ 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 $\pi$ 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, $a_\pi = q_\pi - v_\pi$. [45]

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 $\mathbf{w} \in \mathbb{R}^d$. [45]

An alternative to action-value methods is to approximate the policy itself. *Policy gradient methods* [44] 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 $J(\boldsymbol{\theta})$. As long as $\pi(a|s, \boldsymbol{\theta})$ is differentiable with respect to its parameters, the parameters can be updated with *gradient ascent* in $J$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \hat{J}(\boldsymbol{\theta}_t))$$

Advantages of policy parametrization over action-value methods include stronger convergence guarantees [44] and more flexibility in parametrization [45]. 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.

Function approximation includes important aspects of partial observability. If there is a state variable that is not observable, 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.

#### 2.1.4.3 Challenges in Reinforcement Learning

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.

Another challenge is the design of reeward 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 [45]. Special care has to be taken to ensure that the reward incentivizes the desired behaviour.

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. In such cases, it can be effective to modify the reward to give the agent hints along the way. 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* [30].

In practice, rewards are often designed through trial-and-error. Several iterations of a reward signal are tried until one yields expected and sufficient results.

### 2.1.5 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* [8] 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 $\alpha$:

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

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. [41, 20]

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.

#### 2.1.5.1 Feedforward Neural Network

A feedforward neural network, also known as a multi-layer perceptron (MLP) [20], 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. [41]

#### 2.1.5.2 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. [20]

#### 2.1.5.3 Recurrent Neural Network

Recurrent neural networks (RNNs) extend feedforward 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 feedforward 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. [41]

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 [20]. An architecture that addresses this issue is long short-term memory (LSTM) [26]. 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.

## 2.2 Related Work

To our knowledge, there is no work that considers the exact problem we are looking at. In this section, we present work that considers similar tasks.

### 2.2.1 Deep Reinforcement Learning

As mentioned in Section 2.1.4.2, 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 [46], 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. [36] extend [35] and introduce DQN, which combines deep neural networks with RL and gives birth to the field of deep reinforcement learning (deep RL). DQN approximates the quality function $q(s, a)$ with a CNN, and select actions greedily using only visual input. 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: 32 $8 \times 8$ filters with stride 4, followed by 64 $4 \times 4$ filters with stride 2, followed by 64 filters of size $32 \times 32$ with stride 1. Between each convolutional layer is a ReLU activation function. Then, there is a hidden fully connected layer with a ReLU activation function. The output layer has one output for each valid action.

The DQN architecture sparked great interest and several modifications. Hausknecht and Stone (2017) [22] investigate the effects of adding a recurrent step to DQN in order to tackle POMDPs. They use the same convolutional network as [36], but only use the most recent frame as input and replace the hidden layer with a recurrent LSTM. It is found that the agent is able to integrate information over time and achieves comparable performance to the original DQN agent.

### 2.2.2 Proximal Policy Optimization

Proximal policy optimization algorithms... [43].

**Algorithm 1** Proximal Policy Optimization

**for** iteration $= 1, 2, \ldots$ **do**
  **for** actor $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ for $T$ timesteps
    Compute advantage estimates $\hat{A}_1, \ldots \hat{A}_T$
  **end for**
  Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leqslant NT$
  $\theta_{\text{old}} \leftarrow \theta$
**end for**

### 2.2.3 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 uses deep learning techniques and usually involves a CNN [52].

Although we do not focus on difficult recognition problems in this work, the task we consider bears resemblance to object localization. Object detectors usually use region proposal networks that consider the whole image and return a set of bounding box candidates. The object recognizer is then run on each region proposal.

Usually, images in object detection are passively sampled - they are drawn from some distribution and all are independent. In *active object detection*, images are instead chosen so as to

Caicedo and Lazebnik [10] 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 a trigger 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 trigger 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. [**ghesu_artificial_2016**] present an agent for anatomical landmark detection trained with DRL. Different from [10] 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 anatomical images of the human body.

Chen and Gupta [13] use a spatial memory for context reasoning in object detection...

### 2.2.4 Visual Attention

Visual attention in humans is often split into two phasesis usually split into two phases

[27]
[31]
[34]
Soft attention (Bahdanau [6])...

Partially observable processes, such as the one we consider in this work, can be seens as hard attention problems. By taking actions, the hard attention can be redirected.

### 2.2.5 Coverage Path Planning

The problem we consider in this work shares many characteristics with coverage path planning (CPP) [18]. CPP is the task of determining a path that passes over all points in an area, and appears as a fundamental part of many real-world problems. In the general case, an agent that does not learn to exhaustively search its environment can not be expected to be successful for the visual search task. The CPP is strongly related to the traveling salesman problem (TSP). The unrestricted CPP where there are no obstacles (the "lawnmower problem") is in fact proven to be NP-hard [5]. In our problem, we do not require complete coverage, but just that certain points (those that contain targets) are visited. The shape of the environment is known, as the sensor has limited range of movement. Furthermore, a good agent should also prioritize certain regions which is not part of the CPP task. Although a CPP agent that is able to recognize targets is not optimal, it is a suitable baseline for the task. Specifically, the wavefront algorithm [18] which is suitable for grid-discretized can be used.

### 2.2.6 Visual Navigation

The task we consider bears resemblance to visual navigation [50].

Mnih et al. [33] 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 [36] is used, but with 256 LSTM cells after the final hidden layer.

Mirowski et al. [32]...

Henriques and Vedaldi (2018) [24] use a spatial memory...

Gupta et al. [21] use a latent spatial memory. The They also use a planner that can plan paths given partial information of the environment. This allows the agent to take appearance of visited locations into account when deciding where to look next. The RGB observation is fed through an encoder network that... Planning in this fashion s

Dhiman et al. [16] critically investigate deep RL for navigation. They ask whether DRL 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. However, when trained and tested on differen maps, it cannot do so succesfully. They further find that, with a single decision point whose correct...

Chaplot et al. [11] build on the idea of an explicit memory by including environment semantics...

Zhu et al. [53] create a model for target-driven visual navigation in indooor scenes with DRL. 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 n 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.

A similar work by Ye et al. [49] integrates an object recognition module with a deep reinforcement learning based visual navigation module. They experiment with a set of reward functions and find that constant time penalizing rewards can be problematic and lead to slow convergence. Their experiments make the same assumptions as [**zhu_target_driven**] - the scenes and targets used during testing have all been seen during training.

Several works in visual navigation have placed emphasis on memory representations. [42, 37, 39, 13]

### 2.2.7 Memory Architectures for Deep Reinforcement Learning

1. Frame stacking
2. Recurrent networks
3. Explicit memories [37, 39].

Oh et al. [37] use a differentiable retrieval memory.

Parisotto and Salakhutdinov [39] propose a structured memory...

Anderson et al. [3] emphasize the importance of memory mechanisms that support the construction of rich internal representations environments in navigation agents. Simple agents that are purely reactive and act on the sensory input at the current time step only work for simple tasks. Agumentations like reucrrent update mechanisms add more potential. More advanced memory mechanisms can be important for better navigation. The nature of the internal representation is central to the study of embodied navigation.

### 2.2.8 Benchmarking Environments

- What is contained in an environment (represents a Markov Decision Process).
- What is a good benchmarking environment
- Should list some common environments and explain why they are not satistfactory

### 2.2.9 Inductive Biases, Overfitting and Generalization in Deep Reinforcement Learning

Kirk et al. [29] survey generalization in deep RL.

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 is generally been neglected in deep RL. Training and evaluation stages are typically not separated. Instead, the final return on the training environments is used as a measure of agent performance.

Zhang et al. [51] 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 that the *inductive bias* of the algorithms is 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. [15] 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. [25] 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 removing biases determined with domain knowledge in future research.

Cobbe et al. [14] introduce a benchmark for sample efficiency and generalization in RL. They make use of procedural generalization 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 fix 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.

### 2.2.10 Evaluation of Deep Reinforcement Learning Agents

A problem in state-of-the art RL is reproducibility. There is often non-determinism, both in the methods and environments used. Furthermore, many methods have intrincic variance which can make published results difficult to interpret. This has meant that reproducing state-of-the-art deep RL results is difficult.

Henderson et al. [23] discuss this problem from multiple perspectives. Through experimental analysis, they show that:

- In policy gradient methods, hyperparameters and the choice of network architecture for policy and value function approximation can affect performance significantly. They find 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.
- Rescaling rewards can have a large effect, although it is difficult to predict how.
- Variance between random seeds in stochastic environments affects performance of algorithms, and give learning curves that do not fall within the same distribution. This

suggests that selecting the top $N$ trials or average over a small number of trials $N$ can be misleading. They suggest to compare performance over many different random seeds.

- For certain environments, learning curves can indicate successful optimization but the learned behaviour may not be satisfactory. It is therefore important to not only show returns, but also demonstrations of the learned policy in action.
- 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 codebases with publications. Performance of baseline experiments should also match original baseline publication code.

Due to the unstable nature of RL algorithms, it is often inadequate to just report average return. [23] propose to include confidence intervals when reporting results. Confidence bounds with sample bootstrapping is used to show that PPO is among the more stable algorithms. Various other significance tests...

Finally, [23] make the point that more emphasis should be placed on applying RL algorithms to real-world tasks. Benchmarks environments like ALE [**arcade**] often have no clear winner. 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.

A similar work by Agarwal et al. [1] criticises the heavy use of point estimates of aggregate performance. They show that conclusions drawn from point estimates can be very different from those drawn from more thorough statistical analysis. The popularity of more challenging benchmarks has lead to longer 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 per task. Like [23], they advocate for the use of performance metrics that take uncertainty in results into account. They propose the following set of metrics that better reflect performance across a handful of runs:

- Uncertainty in aggregate performance should be reported through interval estimates via stratified bootstrap confidence intervals.
- Variability in performance across tasks should be reported through performance profiles (score distributions).
- Aggregate metrics for summarizing performance across tasks should be reported through interquartile mean (IQM) across all runs.

Anderson et al. [3] 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

$$\frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{\max(p_i, l_i)} \tag{2.5}$$

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. 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.

Batra et al. [7] revisit the problem of evaluating embodied navigation agents. They note some issues with the SPL metric. It fails to consider the fact that some failures are less of a failure than others. Some failures might in fact be close to reaching the goal while some fail completely. The binary success introduces high variance in average SPL computation. Furthermore, SPL is not particularly suitable for comparison across different datasets, as obtaining a high SPL is more difficult for short paths than for long paths. They suggest that SPL should be replaced by some metric that takes these issues into account. However, to our knowledge such a metric is yet to be proposed and widely adopted.

# 3 Method

In this chapter, the method used is described. Section 3.1 formalizes the problem solved. Section 3.2 details the environment used to evaluate solutions. Section 3.4 describes the baseline learning method. Section 3.3 describes the approach used to solve the problem with a learning agent. Section 3.5 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.

We denote the task by $\langle \mathcal{M}, \mathcal{T}_0 \rangle$, where $\mathcal{M}$ is a POMDP and $\mathcal{T}_0$ is the probability distribution on the initial states. The state is defined by a (Euclidean) space $S \subset \mathbb{R}^d$ which we refer to as the *scene*. At each timestep, the agent observes a subspace $V \subset S$ of the environment which we refer to as the *view*. The actions in $\mathcal{A}$ transform the view. In the scene, there is a set of $N$ targets $T = \{t_0, t_1, \dots t_N | t_i \in S\}$. With a final trigger action, the agent can indicate that there is one or more target in the view. The goal of the agent is to select actions that bring each target into view and indicate that they are visible with the trigger action, while minimizing the number of actions taken. The observations $o \in \Omega$ are tuples $o = \langle x, p \rangle$, where $x \in \mathbb{R}^{3 \times W \times H}$ is an RGB image representing the current view, and $p \in S$ is the position of the agent. If $T \cap V \neq \varnothing$ there are $h = |T \cap V|$ targets in view.

## 3.2 Environments

To train an test an agent for the problem, we use three different environments. The three environments have different characteristics to test the applicability of the evaluated approaches. In each environment there is a scene with a background of distractors and a foreground of targets. As [14, 33], we leverage procedural generation in all environments. The scenes are drawn from some unknown distribution. The appearance of the scenes and the location of targets have some correlation in all environments. This means that the agent should be able to search more efficiently using knowledge from the scene.

For the position part of the observations, we assume the presense of some oracle. In many realistic scenarios this is the case (GPS, pan/tilt, etc.). If how each action moves the agent

is well-defined, we do not need the position at all. We can use relative positions instead of absolute ones. Some of the baselines do not use the position.

The action space is the same in all environments:

$$\mathcal{A} = \{\texttt{UP}, \texttt{DOWN}, \texttt{LEFT}, \texttt{RIGHT}, \texttt{TRIGGER}\},$$

`UP`, `DOWN`, `LEFT`, and `RIGHT` translate the view and `TRIGGER` indicates that a target is in view.

We experiment with three rewards signals. The first reward signal is defined as

$$\mathcal{R}(s_t, a_t) = \begin{cases} 10h & \text{if } a_t = \texttt{TRIGGER} \text{ and a target is in view,} \\ -1 & \text{otherwise.} \end{cases}$$

with $h = |T \cap V|$. We argue that this reward provides a suitable inductive bias for the task at hand. Early experiments show that a constant reward of $r_t = -1$ that simply incentivizes the agent to complete the episode as quickly as possible converge too slowly for large state spaces. The reward for finding a target speeds up training without deviating from the goal of the task - targets should be triggered when in view, but triggers when targets are out of view should be penalized. The constant penalty of $-1$ in all other cases assures that the agent is rewarded for quick episode completion.

In practice, early experiments show that even this reward might be too sparse. To speed up training, we experiment with two extensions to the reward:

$$\mathcal{R}'(s_t, a_t) = \begin{cases} 1 & \text{if } a_t \neq \texttt{TRIGGER} \text{ moves the view closer to the nearest target, and} \\ \mathcal{R}(s_t, a_t) & \text{otherwise.} \end{cases}$$

$$\mathcal{R}''(s_t, a_t) = \begin{cases} 1 & \text{if } a_t \neq \texttt{TRIGGER} \text{ moves the view to a previously unseen subspace, and} \\ \mathcal{R}(s_t, a_t) & \text{otherwise.} \end{cases}$$

These three reward signals are interesting to compare for a few reasons. $R$ does not clearly mediate to the agent what actions are desireable until a target is found. It may therefore lead to slow learning, but it also does not steer away from the goal of finding targets quickly. $R'$ uses the supervised distance between targets and the agents, which is available during training. This is similar to the reward used by [10] and [19]. In addition to speeding up learning, we hypothesize that this reward may help the agent pick up correlations between scene appearance and target probability. However, it can never yield policies that search exhaustively as such actions are never rewarded. It may therefore perform worse during testing where the reward is not available to the agent. It will also not learn to take the shortest paths in the general case, as selecting waypoints greedily does not yield optimal paths. $R''$ strikes a balance between the two other signals by instead incentivizing exploration. This should cause the agent to learn to search the environment exhaustively.

The episode is terminated when all targets have been found, or when 1000 time steps have passed. Terminating episodes early this way is common to speed up training [38].

### 3.2.1 Gaussian Environment

The first environment is the simplest environment. The scene is described by a 256x256 RGB image. The agent observes a 64x64 sub-image at each time step. In the image there are three Gaussian kernels with random positions. The height of the kernel is indicated by a higher intensity in the blue channel. Targets are 1x1 pixels in the red channel. The locations of the targets are randomized weighted by the height of the Gaussian kernels. This means that the more intense the blue channel, the higher the probability of a target. The idea with this environment is to test that the method learns what we want it to learn. There is a clear

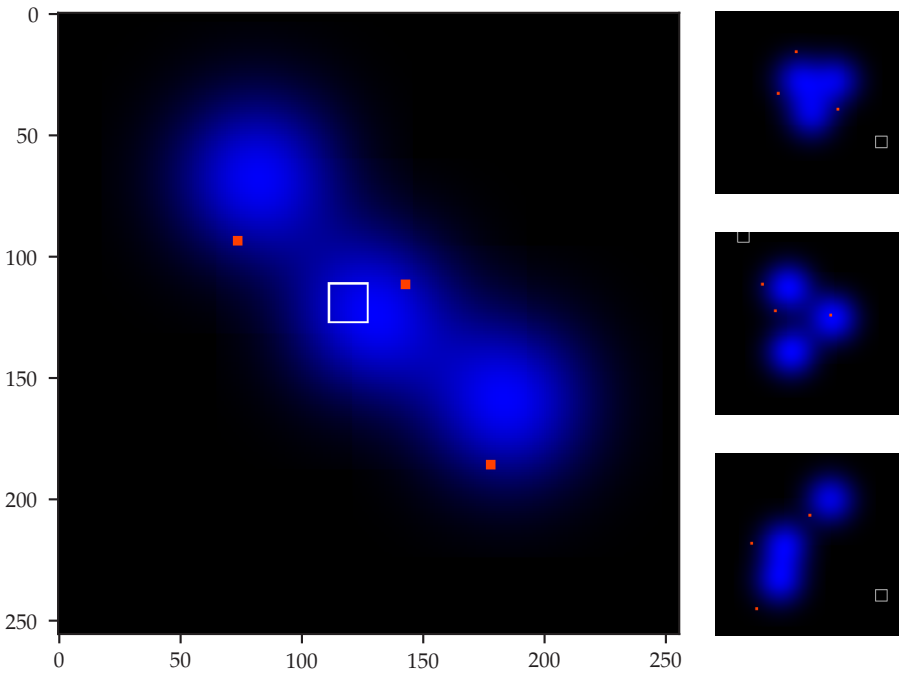


Figure 3.1: Four samples of the first environment. There are three gaussian kernels in the environment, whose height is visualized with the blue channel. There are three targets in the environment, whose location is sampled from the distribution defined by the sum of the three gaussian kernels.

correlation between observations and desirable actions. It is also easy to determine whether the agent acts well in this environment. Our feeling is that this is something that previous similar works has not done.

### 3.2.2 Terrain Environment

The second environment is intended to look like realistic terrain. The scene's appearance is given by a 512x512 RGB image. The agent's view is a 64x64 sub-image. Gradient noise is generated and used as a height map. The height map determines the color of the terrain. The height also correlates to the probability of targets. Specifically, targets are located between shores and mountain bases. There are 10 targets in each scene.

The environment roughly corresponds to a UAV search-and-rescue scenario. It is desirable that a searching agent should learn to not search oceans and lakes. The agent should prioritize searching along the edges of land masses. The appearance of the environment is highly configurable and has high variance. We use this environment for evaluating the generalization capabilities of agents.

### 3.2.3 Camera Environment

The third environment is a three-dimensional version of the second one. The scene

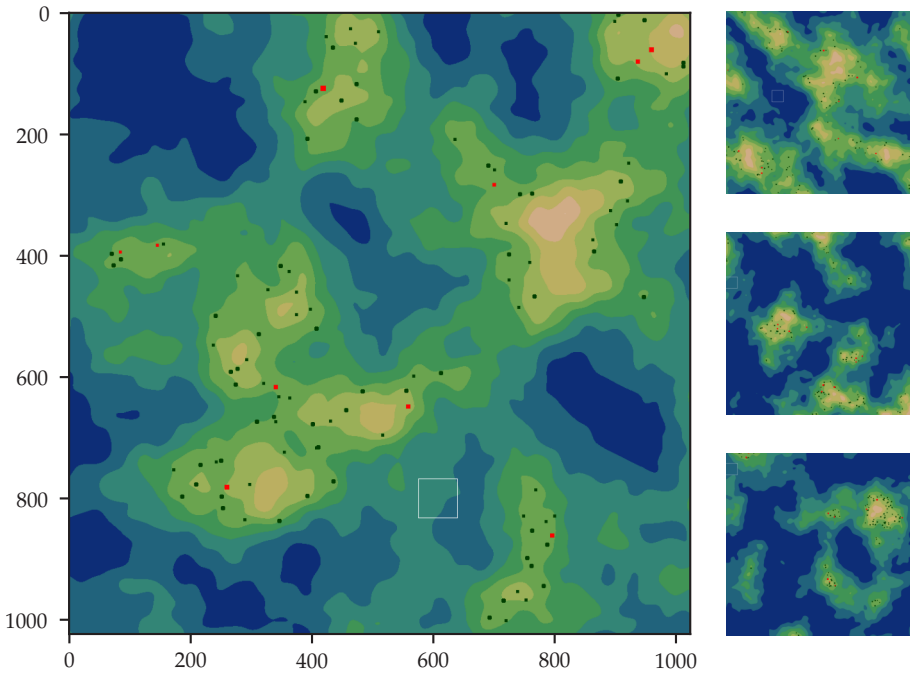


Figure 3.2: Three samples of the terrain environment. Terrain seen from above with red targets scattered along island edges. The white border indicates the agent's current view.

This environment is intended to model more realistic scenarios where the image is more difficult to interpret.

## 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. Considering

Due to time constraints and the advantages described in Section 2.1.4.2, we limit our approaches to policy gradients. Specifically, we use an actor-critic method. The policy and value function are approximated using a multi-headed neural network. The neural network of the agent is split into four parts: A feature extraction network is connected to a recurrent network. The recurrent network is in turn is connected to an actor network head and a critic network head, which approximate the policy and value function respectively.

The architecture of the neural network is presented in Figure X.

The network is trained with PPO [43]. Early experiments show that PPO gives stable learning curves and good sample efficiency, which is in line with results reported by []. Furthermore, we use the same parameters as those outlines in [14]. These are presented in Table 3.1. We find that using many parallell environments stabilizes the training.

[4]

Describe approach once it is fully decided. Will be a memory.

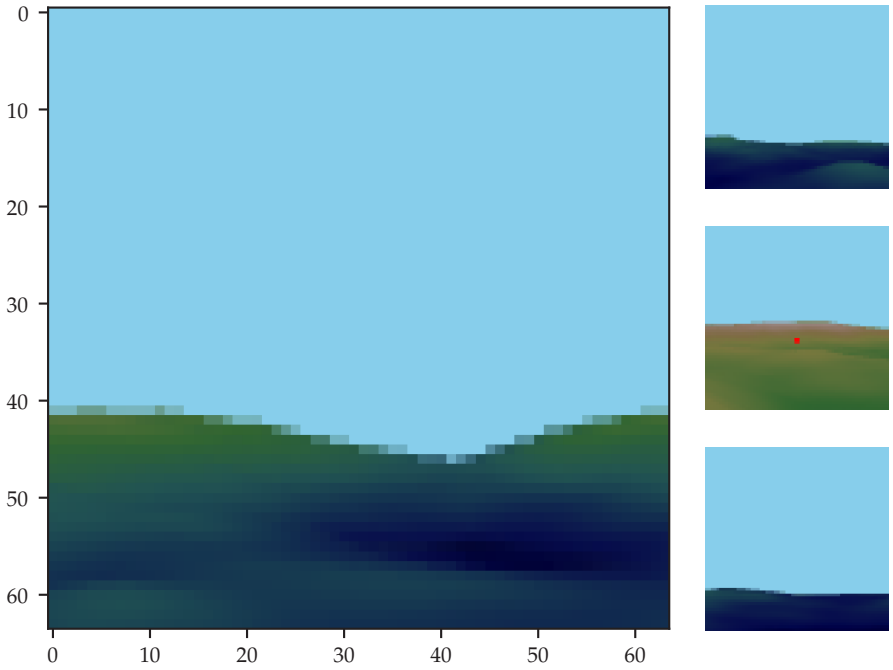


Figure 3.3: Three samples from the camera environment. Terrain seen from a pan-tilt-zoom camera. The pan and tilt of the camera can be adjusted to move the view around. In the mid-right image, a target can be seen.

Table 3.1: Hyperparameters used during training.

| **Parameter** | **Value** |
|---|---|
| $\gamma$ | .999 |
| $\lambda$ | .95 |
| timesteps per rollout | 256 |
| epochs per rollout | 3 |
| minibatches per epoch | 8 |
| entropy bonus | 0.01 |
| clip range | .2 |
| reward normalization | yes |
| learning rate | $5 \times 10^{-4}$ |
| parallell environments | 64 |
| total timesteps | $25 \times 10^{6}$ |

## 3.4 Baselines

We compare our approach to three different well-studied baselines. The first baseline is the agent from [36], which has previously been used as a baseline in .... The agent receives only image observations, and ... The second baseline is a recurrent version of that agent As [32], we use a

This way, we can clearly see the effect memory, image observations and position observations have on performance for the tasks.

We train the baselines with the same algorithm as our approach. We also use the same hyperparameters.

## 3.5 Experiments

To compare our approach to the different baselines, we train and test all agents on all three environments. The agents are trained on the full distribution of environments, We do this for all three reward signals $R$, $R'$ and $R''$. All agents are trained for 25 million time steps. They are then tested on 1000 held out samples from each environment.

For each agent, environment, and reward signal we report the average return and episode length over time during training. During testing, we compare the learning agents to random walk, exhaustive search and a human searcher with prior knowledge of the characteristics of the searched environments. We report the SPL metric 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.

During testing, we increase the maximum episode length from 1000 time steps to 5000 time steps. This is to give more accurate metrics for episodes that do not terminate within 1000 time steps.

Additionally, we conduct experiments to measure the generalization capabilities of the agents. For this, we use the terrain environment only. We test on the same held out levels as before, but vary the number of samples seen during training. The training set size is varied from 100, 1000, 10 000, and 100 000 samples. This is done by limiting the seed pool used to generate the environments. This way, we can get a sense of how much data and simulation is required to apply the approach to real-world tasks. For each training set size, we train the agents until convergence. This is the same approach as in [14]. Once again, we compare all three reward signals.

Following the recommendations of [23] and [1], we report confidence intervals across a handful of seeds. We run all experiments across 3 different runs, and use the aggregate metrics proposed by [1].

## 3.6 Implementation

The environment is implemented with Gym [9]. The agent is implemented and RL algorithms are implemented with PyTorch [40] for automatic differentiation of the computation graphs.

Proximal policy optimization was implemented following the official implementation by OpenAI. Some necessary modifications were made to allow for recurrent policies.

All experiments are conducted on an Intel Core i9-10900X CPU and an NVIDIA GeForce RTX 2080 Ti GPU.

# 4 Results

# 5 Discussion

This chapter contains the following sub-headings.

## 5.1 Results

## 5.2 Method

## 5.3 The work in a wider context

# 6 Conclusion

# Bibliography


[1] 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 15, 22).

[2] 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).

[3] 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, 15).

[4] 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 page 20).

[5] Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. "Approximation algorithms for lawn mowing and milling". In: *Computational Geometry* 17.1 (Oct. 2000), pp. 25–50. ISSN: 0925-7721. DOI: 10.1016/S0925-7721(00)00015-8 (Cited on page 12).

[6] 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 12).

[7] 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 16).

[8] 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 8).

[9] 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 22).

[10] 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 11, 18).

[11] 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 page 12).

[12] 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 page 5).

[13] 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 pages 11, 13).

[14] 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 14, 17, 20, 22).

[15] 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 page 14).

[16] 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 page 12).

[17] 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 2, 6).

[18] 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 12).

[19] Florin-Cristian Ghesu, Bogdan Georgescu, Yefeng Zheng, Sasa Grbic, Andreas Maier, Joachim Hornegger, and Dorin Comaniciu. "Multi-Scale Deep Reinforcement Learning for Real-Time 3D-Landmark Detection in CT Scans". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41.1 (Jan. 1, 2019), pp. 176–189. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2017.2782687. URL: https://ieeexplore.ieee.org/document/8187667/ (visited on 02/03/2022) (Cited on page 18).

[20] 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 9, 10).

[21] 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 page 12).

[22] 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 page 10).

[23] 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 14, 15, 22).

[24] 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 page 12).

[25] 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 page 14).

[26] 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 page 10).

[27] 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 page 12).

[28] 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 6, 7).

[29] 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 page 13).

[30] 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 8).

[31] 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 page 12).

[32] 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 12, 22).

[33] 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 12, 17).

[34] 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 page 12).

[35] 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 page 10).

[36] 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 10, 12, 22).

[37] 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 page 13).

[38] 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).

[39] 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 page 13).

[40] 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 22).

[41] 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 9, 10).

[42] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. "Semi-parametric Topological Memory for Navigation". In: *arXiv:1803.00653 [cs]* (Mar. 2018). arXiv: 1803.00653. URL: http://arxiv.org/abs/1803.00653 (Cited on page 13).

[43] 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 11, 20).

[44] 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 pages 7, 8).

[45] 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 6–8).

[46] Gerald Tesauro et al. "Temporal difference learning and TD-Gammon". In: *Communications of the ACM* 38.3 (1995), pp. 58–68 (Cited on page 10).

[47] Jeremy M. Wolfe. "Guided Search 6.0: An updated model of visual search". In: *Psychonomic Bulletin & Review* 28.4 (Aug. 2021). 29 citations (Crossref) [2022-03-02], pp. 1060–1092. ISSN: 1531-5320. DOI: 10.3758/s13423-020-01859-9 (Cited on pages 2, 6).

[48] 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 page 2).

[49] 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 13).

[50] Fanyu Zeng, Chen Wang, and Shuzhi Sam Ge. "A Survey on Visual Navigation for Artificial Agents With Deep Reinforcement Learning". In: *IEEE Access* 8 (2020), pp. 135426–135442. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3011438 (Cited on page 12).

[51] 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 page 13).

[52] 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).

[53] 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 page 12).