

Deep Reinforcement Learning Applied to an Image-Based Sensor Control Task

Rickard Eriksson

Master of Science Thesis in Electrical Engineering

Deep Reinforcement Learning Applied to an Image-Based Sensor Control Task

Rickard Eriksson

LiTH-ISY-EX--21/5410--SE

Supervisor: **Joakim Argillander**
ISY, Linköpings universitet
Fredrik Bissmarck
FOI

Examiner: **Ingemar Ragnemalm**
ISY, Linköpings universitet

*Division of Information Coding
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden*

Copyright © 2021 Rickard Eriksson

Abstract

An intelligent sensor system has the potential of providing its operator with relevant information, lowering the risk of human errors, and easing the operator's workload. One way of creating such a system is by using reinforcement learning, and this thesis studies how reinforcement learning can be applied to a simple sensor control task within a detailed 3D rendered environment. The studied agent controls a stationary camera (pan, tilt, zoom) and has the task of finding stationary targets in its surrounding environment. The agent is end-to-end, meaning that it only uses its sensory input, in this case images, to derive its actions. The aim was to study how an agent using a simple neural network performs on the given task and whether behavior cloning can be used to improve the agent's performance.

The best-performing agents in this thesis developed a behavior of rotating until a target came into their view. Then they directed their camera to place the target at the image center. The performance of these agents was not perfect, their movement contained quite a bit of randomness and sometimes they failed their task. But even though the performance was not perfect, the results were positive since the developed behavior would be able to solve the task efficiently given that it is refined. This indicates that the problem is solvable using methods similar to ours. The best agent using behavior cloning performed on par with the best agent that did not use behavior cloning. Therefore, behavior cloning did not lead to improved performance.

Acknowledgments

Special thanks to Fredrik Bissmarck, my supervisor at FOI, for his support and valuable insights throughout the thesis. I am also grateful to Johan Karlsson, Viktor Deleskog, and Fredrik Näsström at FOI for their investment in my thesis and great help whenever problems and questions arose.

Further, I would like to thank Joakim Argillander, my supervisor at LiU, for providing great feedback during the writing process, as well as Ingemar Ragnemalm, my examiner, for his guidance and help.

Lastly, I would like to thank the other master's thesis students at FOI who have provided good company during coffee breaks.

Linköping, June 2021
Rickard Eriksson

Contents

Notation	ix
1 Introduction	1
1.1 Background and Purpose	2
1.2 Problem Statement	2
1.3 Related Work	2
1.4 Delimitations	3
2 Theory	5
2.1 Reinforcement Learning Introduction	5
2.1.1 Concepts	5
2.1.2 Q-learning	8
2.2 Function Approximation	9
2.2.1 Feedforward Neural Networks	10
2.2.2 Convolutional Neural Network	10
2.3 Proximal Policy Optimization	12
2.3.1 Policy Gradient	12
2.3.2 Clipping	13
2.3.3 Algorithm	13
2.4 Behavior Cloning	15
3 Method	17
3.1 Design	17
3.1.1 Agent	17
3.1.2 Environment	18
3.1.3 Resetting the Environment	19
3.1.4 Goal and Reward Signal	19
3.1.5 Implementation	21
3.2 Behavior Cloning	22
3.3 Performance	22
3.4 Experiments	23
4 Results	25

4.1	Agent Training	25
4.2	Agent Behavior	27
5	Discussion	31
5.1	Results	31
5.1.1	Behavior	31
5.1.2	Behavior Cloning	32
5.2	Method	33
5.2.1	Sample Efficiency	33
5.2.2	Problem Difficulty	33
5.2.3	Neural Network	34
5.2.4	Performance Metric and Reward Signal	34
5.2.5	Reliability and Validity	34
5.2.6	Source Criticism	35
5.3	Future work	35
5.4	Work in a Wider Context	37
6	Conclusions	39
	Bibliography	41

Notation

NOTATION

Notation	Meaning
t	Time step
T	Last time step in episode
s	State
a	Action
r	Reward
s_t	State at time step t
a_t	Action at time step t
r_t	Reward at time step t
S_t	State at time step t , stochastic
A_t	Action at time step t , stochastic or advantage
R_t	Reward at time step t , stochastic
G_t	Return at time step t , stochastic
π	Policy
π_*	Optimal policy
$\pi(a s)$	Probability of taking action a in state s under policy π
$V_\pi(s)$	State-value function, stochastic
$Q_\pi(s, a)$	Action-value function, stochastic
α	Learning rate
γ	Discount rate
ϵ	Exploration factor or clipping range
x	Input to neural network, vector
y	Output of neural network
θ	Weights for neural network
$\theta_{i,j}$	Weight between neuron i and j
$L(\theta)$	Loss
$\Pr(X)$	Probability of X
$\mathbb{E}\{X\}$	Expected value of X
$\hat{E}[X]$	Batch average of X
$J(\theta)$	Performance measure
$\overline{\nabla J(\theta)}$	Estimate of performance measure's gradient
\hat{A}_t	Estimated advantage at time step t
$S_\pi(s)$	Entropy

1

Introduction

"A behavior that is followed by a satisfying state of affairs—a reward—is more likely to increase in frequency than a behavior not followed by a reward." [1, p.7]

This is how Jeanne Ellis Ormrod describes one of the core principles of learning in her book *Human learning* [1]. It is a principle that we are all familiar with, and a classic example would be teaching a dog to perform a trick by rewarding it with food. In the field of psychology, this principle of linking rewards to behavior is known as reinforcement [1]. This is what has inspired the name of reinforcement learning, a field within machine learning [2].

Reinforcement learning is a field that aims to create intelligent agents through a process of trial and error. The agent is the learner, and after it performs an action it is given a reward depending on whether that action was good or bad. A close connection can be seen between this and reinforcement in psychology. By trying out different actions and observing the reward it receives, the agent can obtain a good behavior from trying to maximize this reward. When and how much reward the agent receives is defined by the so-called reward signal, and the reward signal is therefore also what defines the agent's goal. [3]

One of the benefits of reinforcement learning is that learning is centered around this reward signal. This means that the engineer designing the system does not have to explicitly define the agent's behavior, which can be a difficult task. Instead, the engineer only has to decide when the agent should be rewarded for its behavior. [3] This is not to say that the task of designing a reward signal is easy. It often requires great care to design a reward signal that results in a well-performing agent, but it is in many cases better than the alternative of explicitly defining the agent's behavior.

1.1 Background and Purpose

Using automated systems has many benefits, including increased effectivity, increased safety, and lowered workload. This is no different in the field of reconnaissance, where an intelligent sensor system can be of great value to its operator. An intelligent sensor system can provide its operator with relevant information, lower the risk of human errors, and ease the workload of its operator.

This thesis aims to investigate how reinforcement learning can be applied to a sensor control task within a 3D rendered environment. The agent should be end-to-end, meaning that it is able to learn directly from its sensory input and, based on this, decide its actions. Therefore, it needs to be able to derive a representation of its environment based on this sensory input, which in this case is images. Being able to derive a representation of its environment is a valuable attribute since it removes the need for hand-crafted features.

The agent in this thesis controls a stationary camera (pan, tilt, zoom) and has the goal of finding stationary targets in its surrounding environment. This setup is explained in further detail in Chapter 3.

1.2 Problem Statement

The thesis aims to investigate the following questions:

1. An agent, using images, has the task of finding stationary targets by controlling a stationary camera (pan, tilt, zoom). How does the agent, using a simple neural network, perform on this task?
2. How does the performance and behavior of the agent described above change when pre-trained with behavior cloning, an imitation learning method?

How well an agent performs is defined by how much reward it obtains and through visual examination of its behavior.

1.3 Related Work

Since the introduction of deep learning, advancements have been made within the field of reinforcement learning. Examples include agents capable of beating professionals at the Chinese board game Go [4, 5], playing video games at a competitive level [6, 7], performing different control tasks [8], and more [9]. One influential publication published in 2015 by Mnih et al. [10] presented an agent capable of playing different Atari 2600 games. It was not only able to play these games at a human-level performance, but it was able to do this using only images. As opposed to requiring hand-crafted features, the agent could derive a representation of its environment using these images. This is beneficial since, by deriving its own representation of the environment, the agent is not bound to any hand-crafted features and is therefore not bound to a specific environment either.

The same agent can be applied to other similar environments without requiring its design to be altered, as is seen in [10] where the same agent was able to be trained and perform on several different games. [10]

There are other examples of image-based agents other than the ones playing Atari games. An example of this would be the agents using ViZDoom, a platform based on the game Doom, that allows for the development of image-based agents [11]. Examples of such agents include one that was taught to collect health kits [11] or one that was taught to play the deathmatch game mode [12]. Further, reinforcement learning has been successfully applied to control tasks such as dexterous manipulation and legged locomotion using images [13]. There has also been research on using images and reinforcement learning to teach agents to navigate indoors [14].

1.4 Delimitations

Due to the limited time and computing resources available (an experiment can take over a week), some delimitations have to be made. Different neural network designs will not be investigated due to time constraints. For the same reason, hyperparameter tuning, which is the search for optimal hyperparameters, will not be performed. Since the size of the neural network affects training times, the examined network will be kept small. The network examined will consist of 3 convolutional layers and 2 fully connected layers. The size of images also affects training times and will therefore be constrained to a size of 640×480 pixels. Further, all targets in the environment will be stationary.

2

Theory

In this chapter, concepts central to the field of reinforcement learning are presented as well as the algorithm Proximal Policy Optimization (PPO). Also, the basics of feedforward neural networks and convolutional neural networks are covered.

2.1 Reinforcement Learning Introduction

Reinforcement learning is a field in machine learning that aims to create intelligent agents by letting them interact with an environment that returns a reward based on how well they do. By exploring different actions and observing the reward, the agent alters its decision-making process to try and maximize the accumulated reward. Through this process of trial and error, the agent can learn to solve tasks intelligently. [2]

2.1.1 Concepts

This subchapter presents some of the key concepts in reinforcement learning.

Environment

The environment is where the agent acts and learns from experience. The environment provides the agent with samples containing information about its current state S_t , as well as the reward R_t , through what is called observations [3]. An example can be made of the game "tic-tac-toe". The state of this environment would be the board state, in other words, what each position contains (X, O, or

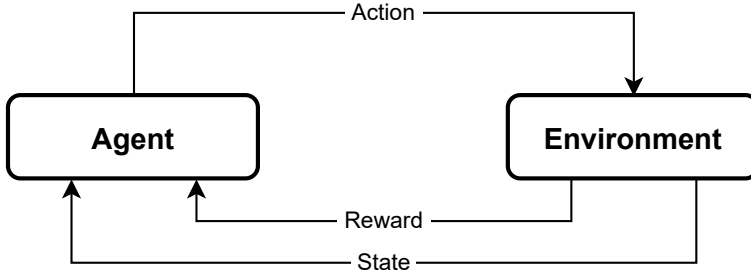


Figure 2.1: Standard agent-environment interaction. The agent performs an action on the environment which is updated accordingly. The environment returns a reward as well as the updated state.

empty). A reward could be as simple as +1 for winning the game and +0 for losing.

An environment can be either fully or partially observable. In a fully observable environment, the agent has full knowledge of the state it is in, and in a partially observable environment, the agent may not be aware of its exact state. This could be due to factors such as sensor noise. [9]

Agent

The agent is the learner, decision-maker, and actor of the reinforcement learning problem. The agent and environment interact continually with each other. Based on the state received from the environment, the agent chooses an action to perform. After the action has been performed the environment is updated accordingly and returns a reward. Based on this reward, the agent updates its policy/value function. The agent then chooses a new action using the environment's new state, and the interaction loop repeats. Interactions are performed in a sequence of time steps $t = 0, 1, 2, \dots, T$ where a time step denotes one whole interaction loop. The agent-interaction loop can be seen in Figure 2.1. [3]

Policy

The policy π is the decision-making part of an agent. It is therefore also what defines the agent's behavior. The policy is a mapping between actions and states, and it informs the agent about what action to take given that it is in a specific state. The design of a policy can vary wildly, from being a lookup-table to complex computations, and many times, the policy is stochastic in which $\pi(a|s)$ denotes the probability to take action a in state s . The optimal policy, π_* , is the policy that always chooses the action corresponding to the highest estimated reward. [3]

Reward Signal and Return

The reward signal R_t is the reward obtained by the agent at time step t . The agent's goal is to maximize the cumulative reward, also known as the return G_t . A simple return can be defined as the sum

$$G_t = R_{t+1} + R_{t+2} + \dots + R_T = \sum_{k=0}^{T-t-1} R_{t+1+k} \quad (2.1)$$

where T is the final time step. [3] Oftentimes the return includes a discount rate that suppresses the impact of future rewards. In that case the return is written as

$$G_{t,\text{discounted}} = \sum_{k=0}^{T-t-1} \gamma^k R_{t+1+k} \quad (2.2)$$

where $0 < \gamma \leq 1$ is the discount rate. There are several reasons to suppress future rewards, examples being if the problem is continuous and never ends or to account for uncertainty in future predictions. [2]

Value Function

The state-value function, $V_\pi(s)$, is the expected future reward from ending up in state s given that we follow policy π . It is a measure of how good it is to be in a given state, and the value function can be written as

$$V_\pi(s) = \mathbb{E}_\pi \{r_t | s_t = s\} = \mathbb{E}_\pi \left\{ \left(\sum_{k=0}^{T-t-1} \gamma^k r_{t+1+k} \right) | s_t = s \right\} \quad (2.3)$$

where γ is the discount rate, s_t is the state at time step t , r_t is the reward at time step t , and \mathbb{E} is the expected value. [2]

In Figure 2.2 a grid-world environment is shown where the agent, in this case a robot, has the task of getting to the finish by moving left, right, up, and down. The agent is rewarded for reaching the goal, but below the goal there is a pit. If the agent falls down this pit, it is penalized. Each grid displays a value that is the state-value function for that state. As seen in Figure 2.2, the state-value function is higher close to the finish since these are good states to be in, and lower close to the pit.

Episode

The interaction between agent and environment shown in Figure 2.1 can be split into subsequences called episodes. An episode ends when the agent reaches a terminal state, and afterward, the environment is reset. In the grid-world example of Figure 2.2 the episode would be the sequence of time steps between the agent starting and reaching the finish. [3]




0.48	0.55	0.64	0.78	0.85
0.41		0.72	0.86	 1
0.35		0.65	0.58	 -1
	0.43	0.52	0.44	0.21

Figure 2.2: A grid-world environment. Each grid is a state and the displayed value is the value of the state-value function $V_\pi(s)$. The agent is rewarded for reaching the finish (marked by two flags) and punished for falling down the pit (marked by a skull).

Exploration

To be able to find a good solution to the given task, the agent has to explore. If it were to strictly follow the optimal policy, it could easily get stuck in a subpar solution. By forcing the agent to explore it can find new and better solutions. An example of an exploring policy is the ϵ -greedy policy, in which there is a ϵ chance for the agent to take a random action and a $1 - \epsilon$ chance for the agent to pick an action based on its policy. [3]

2.1.2 Q-learning

One of the most well-known reinforcement learning algorithms is the Q-learning algorithm. Q-learning makes use of the action-value function $Q_\pi(s, a)$ which is a measure of how good it is to take action a in state s . The action-value function can be written as

$$Q_\pi(s, a) = \mathbb{E}_\pi\{r_t | s_t = s, a_t = a\} = \mathbb{E}_\pi\left\{\left(\sum_{k=0}^{T-t-1} \gamma^k r_{t+1+k}\right) | s_t = s, a_t = a\right\} \quad (2.4)$$

and is similar to the state-value function $V_\pi(s)$ except that it, in addition to the state, also considers the action. [2] The policy used in Q-learning is an optimal ϵ -greedy policy where the agent has an ϵ chance of taking a random action and an $1 - \epsilon$ chance of taking the optimal action. The optimal action is the one that maximizes the action-value function $\max_a Q_\pi(s, a)$. [3]

The Q-learning algorithm starts by initializing $Q_\pi(s, a)$ with arbitrary values. Thereafter, for each time step, an action is chosen based on the agent's policy. The agent performs the action and observes the reward and a new state. Thereafter, $Q_\pi(s, a)$ is updated according to

$$Q_\pi(S_t, A_t) \leftarrow Q_\pi(S_t, A_t) + \alpha [R_{t+1} + \gamma \max_a Q_\pi(S_{t+1}, a) - Q_\pi(S_t, A_t)] \quad (2.5)$$

where α is the learning rate. The learning rate dictates to what degree the old value should be influenced by the update. This process is repeated until the episode is done. When the episode is done, the environment is reset and the process starts over. The outline of the Q-learning algorithm can be seen in Algorithm 1 below. [3]

Algorithm 1: Q-learning

```

Initialize  $Q_\pi(s, a)$  with random values;
for each episode do
    Reset environment;
    while not done do
        Choose action  $A$  based on state  $S$  using policy  $\pi$ ;
        Perform action  $A$  and observe  $R, S'$ ;
         $Q_\pi(S, A) \leftarrow Q_\pi(S, A) + \alpha [R + \gamma \max_a Q_\pi(S', a) - Q_\pi(S, A)]$ ;
         $S \leftarrow S'$ ;
        if episode end then
            done;
        end
    end
end

```

By running the Q-learning algorithm, the action-value function should eventually get to a state where it allows the agent to solve the given task in a satisfactory way.

2.2 Function Approximation

In many cases, the state-value and action-value functions are tables over all possible states (and actions). This works when the state space is small, such as the grid-world in Figure 2.2 with 18 states (each grid), but for most real-world tasks the state space is far too large for this to be possible. In this project images are used, and this is an example of when the state space becomes too large. If we have an image that is 640×480 pixels large, with each pixel containing an RGB value that ranges from 0 – 255, we have a state space with $256^{3(640 \times 480)} \approx 9 \times 10^{2219433}$ possible states. That is too many states to fit into a table, and using the standard Q-learning algorithm would not be feasible. A solution to this is to use function approximation methods, such as neural networks, to approximate the value function or policy instead. [3]

2.2.1 Feedforward Neural Networks

Feedforward neural networks are an example of function approximation methods. Given a function $y = f^*(x)$ that defines a mapping from input x to output y , the network tries to create an approximation. The approximation is made via the mapping of $y = f(x; \theta)$ where θ are parameters that have to be learned, also known as the weights of the network. [15]

The main component of a neural network is the neuron, see Figure 2.3. The input x of a neuron is multiplied by the weights θ and summed up. This becomes the neuron's output which is then sent to an activation function. The activation function has the task of adding non-linearity, which is needed for the network to be able to approximate arbitrary functions. There exists a wide range of different activation functions, and ReLU, rectified linear units, is the one that is most commonly used in modern neural networks. The ReLU activation function is defined as

$$g(z) = \max(z, 0) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2.6)$$

where z is the summed output of the neuron. [15] By arranging the neurons into layers and connecting them, a neural network can be created, such as the one in Figure 2.4. A network like this has the possibility of approximating non-linear functions given that its weights are tuned correctly. It could, as an example, be used to approximate the value function or policy of an agent. In that case, the state s is used as input to the network.

Tuning the weights of a network (training) can be done via gradient descent. The used loss function differs depending on application, but a simple and common example is the squared loss $L = (y - \hat{y})^2$ where \hat{y} is the network's prediction and y the true value. Based on this loss, the weights are updated by gradient descent

$$\theta_{i,j} \leftarrow \theta_{i,j} - \alpha \frac{\partial L}{\partial \theta_{i,j}} \quad (2.7)$$

where α is the learning rate which controls the size of the update, and $\theta_{i,j}$ is the weight between neuron i and neuron j . [9]

2.2.2 Convolutional Neural Network

In the example of a feedforward neural network, the input x is given as a vector of data. But what if the input is an image? One idea is to vectorize the image and reorder it in such a way that it consists of one single row of pixel data. This then becomes the input x . However, doing this will result in the loss of spatial information, which is important in images and should be preserved. To make use of local spatial information, a convolutional neural network (CNN) can be used. In a CNN, the bottom layers are convolutional layers. These layers use images as input and apply a kernel via convolution. The result of this operation is a new image which is then fed into the next convolutional layer. In a convolutional layer,

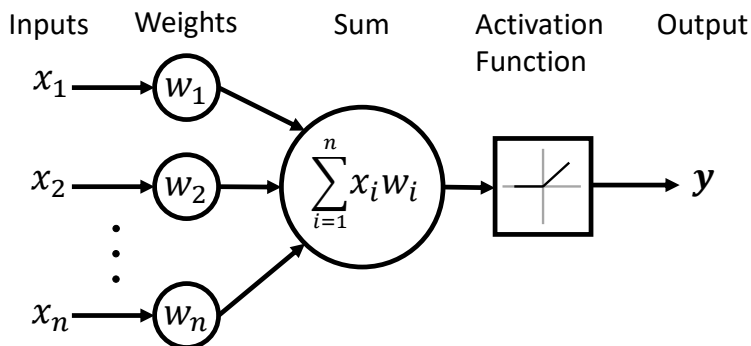


Figure 2.3: An artificial neuron. The inputs are multiplied by their corresponding weights and summed up. This sum is then sent to the activation function which generates the neurons output.

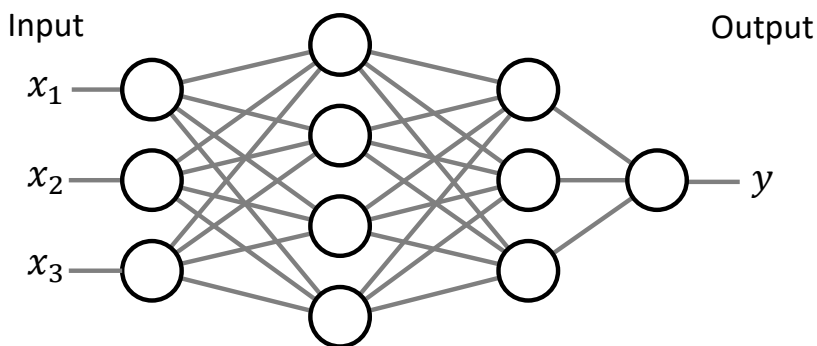


Figure 2.4: A feedforward neural network. Each node denotes an artificial neuron, and a column of neurons is called a layer.

there are no weights between neurons. Instead, the weights of a convolutional layer are the elements of the kernel. [9]

2.3 Proximal Policy Optimization

Proximal policy optimization, also known as PPO [16], is a reinforcement learning algorithm that has gotten lots of attention in recent years. It has been used to teach agents to play complex video games at a professional level [6], solve control tasks [8] and much more. PPO has become popular because it is an algorithm that is data-efficient, robust, and easier to understand than similar algorithms such as trust region policy optimization [16, 17].

This section introduces the methods PPO is based on, policy gradient methods, followed by covering the clipping that is done to avoid destructively large updates to the policy. Lastly, the algorithm itself is presented.

2.3.1 Policy Gradient

The policy is a mapping between actions and states, and it informs the agent about what action to take given that it is in a specific state. Policy gradient methods are a set of reinforcement learning algorithms that learn the parametrized policy $\pi(a|s, \theta)$. θ is the the policy's parameter vector and given that a neural network is used to approximate the policy, θ is the weights of that neural network. The parametrized policy describes the probability of taking action a given that the agent is in state s with parameters θ at time step t

$$\pi(a|s, \theta) = \Pr\{A_t = a | S_t = s, \theta_t = \theta\}. \quad (2.8)$$

Policy gradient methods aim to maximize a performance measure $J(\theta)$ [3] which can also be called the objective or loss [16]. The updates to θ are done via gradient ascent

$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)} \quad (2.9)$$

where $\widehat{\nabla J(\theta_t)}$ is an estimate of the performance measure's gradient [3]. PPO is based on a subgroup of policy gradient methods called actor-critic which, in addition to the policy, learns the value function. A frequently used gradient estimator for these is

$$\widehat{\nabla J(\theta_t)} = \hat{E}_t[\nabla_{\theta} \log \pi_{\theta}(a_t|s_t) \hat{A}_t] \quad (2.10)$$

where \hat{E}_t denotes the batch average and \hat{A}_t is the estimated advantage function. The estimated advantage function is calculated as

$$\hat{A}_t = \delta_t + (\gamma \lambda) \delta_{t+1} + \dots + (\gamma \lambda)^{T-t+1} \delta_{T-1} \quad (2.11)$$

where

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \quad (2.12)$$

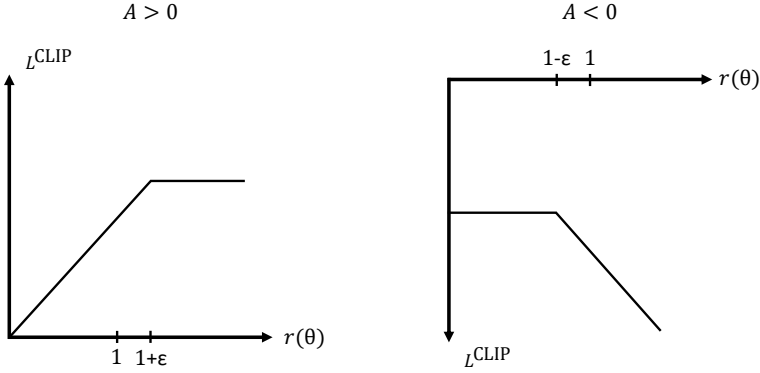


Figure 2.5: The clipping done in PPO to ensure L^{CLIP} is pessimistic. The left plot shows the clipping for negative advantages and the right plot shows the clipping for positive advantages. If $A = 0$ then $L^{\text{CLIP}} = 0$.

and λ is a coefficient. From the gradient estimator in equation 2.10, the objective can be derived as

$$L^{\text{PG}}(\theta) = \widehat{J}(\theta) = \hat{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t] \quad (2.13)$$

which is called the policy gradient loss. [16]

2.3.2 Clipping

In PPO policy updates are performed several times using the same trajectory. If not handled correctly, this can lead to destructively large updates and is the reason the policy gradient loss, $L^{\text{PG}}(\theta)$, is not used. To avoid large updates, a clipped objective is used instead. The clipped objective is a pessimistic estimate and is formulated as

$$L^{\text{CLIP}}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (2.14)$$

where ϵ defines the clipping range and $r_t(\theta)$ is the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta, \text{old}}(a_t|s_t)}. \quad (2.15)$$

The clip function makes sure that the probability ratio does not go outside of the bounds $1 \pm \epsilon$ and the min function makes sure that the estimate is pessimistic as it chooses the lowest value. [16]

2.3.3 Algorithm

When using neural networks to approximate both the policy and value function it is common to use a shared network instead of two separate networks. The

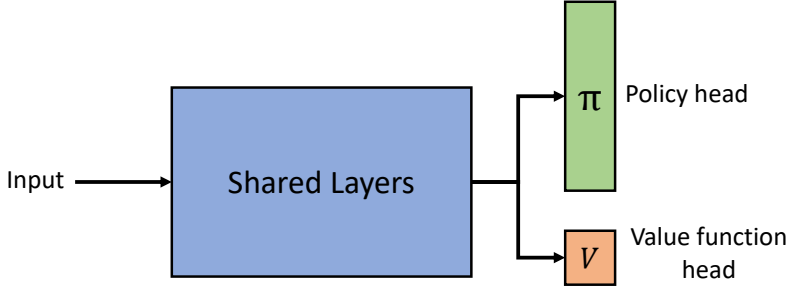


Figure 2.6: A shared network. The approximation for both the policy and value function uses the same bottom layers (the shared layers). Only the top layers of the network are specific to the policy/value function and are known as heads.

bottom layers of a shared network are shared, meaning that the same layers are used for both policy and value approximations. The top layers, however, are split into separate layers that are tailored for the specific approximation. See Figure 2.6. When a shared network is used, the objective needs to be a combination of $L^{\text{CLIP}}(\theta)$ and the value function loss

$$L^{\text{VF}} = \left(V_{\theta}(s_t) - V_t^{\text{target}} \right)^2. \quad (2.16)$$

A term called entropy bonus, $S_{\pi_{\theta}}(s_t)$, is also added to the loss function and it is used to promote exploration by the agent. [16] The entropy bonus is the Shannon entropy and is calculated by

$$S_{\pi_{\theta}}(s_t) = - \sum_{i=1}^n \pi_{\theta}(a_i|s_t) \log(\pi_{\theta}(a_i|s_t)). \quad (2.17)$$

where $\pi_{\theta}(a|s)$ is the probability of taking action a in state s following policy π_{θ} [18]. Combining these losses, we end up with

$$L_t^{\text{CLIP} + \text{VF} + \text{S}}(\theta) = \hat{E}_t[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S_{\pi_{\theta}}(s_t)] \quad (2.18)$$

where c_1 and c_2 are coefficients. c_1 is known as the value function coefficient and c_2 is known as the entropy coefficient.

The PPO algorithm, see Algorithm 2, starts by initializing the agent's weights θ . Thereafter, T time steps of data are sampled using policy $\pi_{\theta, \text{old}}$, creating what is known as a trajectory. The advantages are calculated and after the data has been collected, the policy is updated by optimizing $L^{\text{CLIP} + \text{VF} + \text{S}}(\theta)$ with respect to θ . The optimization is performed several times for the same trajectory using stochastic gradient ascent. These optimization steps that use the same trajectory are called epochs. [16]

Algorithm 2: PPO

```

Initialize  $\theta$ ;
for iteration  $i = 0, 1, \dots$  do
  for time step  $t = 0, 1, \dots, T$  do
    Sample time step with policy  $\pi_{\theta, \text{old}}$ ;
    Calculate advantage  $\hat{A}_t$ ;
  end
  for epoch  $k = 0, 1, \dots, K$  do
    Optimize  $L^{\text{CLIP+VF+S}}$  with respect to  $\theta$ ;
    Update  $\theta$ ;
  end
end

```

2.4 Behavior Cloning

Imitation learning methods are a group of methods that make use of demonstrations in order to learn [9, 19]. Behavior cloning is one of these methods, and the goal of behavior cloning is for an agent to copy the behavior of an expert based on the given demonstrations. Demonstrations are generated by saving the experiences of an expert interacting with the environment, and the information saved in these experiences can differ depending on implementation. [19] Examples of saved data would be the state s_t , expert action e_t , reward r_t , and so on. The expert can be an algorithm designed to solve the same task as the agent, or a human that is given the possibility to manually control the agent. The loss function used in this project is based on the probability that the agent's policy picks the same action as the expert. This can be written as

$$L^{\text{BC}}(a, \pi) = -\log(\Pr(\pi(s) = e)) \quad (2.19)$$

where e is the expert action and $\pi(s)$ is the action chosen by the agent.

The aim of using behavior cloning is to pre-train the agent and give it a good base behavior that allows it to explore efficiently. The agent should then continue training with a "normal" reinforcement learning algorithm, such as PPO, with the hopes that the induced base behavior improves the training. [19]

3

Method

In this chapter, the design of both agent and environment is presented. The implementation of behavior cloning is also covered, as well as the different setups of agents that were trained.

3.1 Design

This section aims to cover how both the agent and environment are designed. The goal of the agent is to find stationary targets by controlling a stationary camera (pan, tilt, zoom), and the agent is only allowed to use images as input.

3.1.1 Agent

When it comes to reinforcement learning algorithms there are many to choose from. The agents in this project use Proximal Policy Optimization (PPO) since it is an algorithm that has proven to achieve good performance on a wide variety of tasks [6, 8, 16] as well as being robust. Being a robust algorithm means that it performs well on many tasks without being dependent on the use of hyperparameter tuning, the search for optimal hyperparameters [16]. In Chapter 2.3 the PPO algorithm is explained in more detail.

The agent's neural network model is the same as in [10] and consists of three convolutional layers and two fully connected layers. This network design is quite small, and using a small network was a conscious choice since network size affects training times. The network design was deemed suitable since there exists other image-based agents that use networks of similar size and design [10–12]. In PPO, both the value function and policy need to be approximated, and therefore, a shared network was used. This means that the same base network is utilized for both approximations. In Tables 3.1-3.2 and Figure 3.1 the network is described in more detail.

Layer	Conv. 1	Conv. 2	Conv. 3
Input dimension	$480 \times 640 \times 3$	$119 \times 159 \times 32$	$58 \times 78 \times 64$
Kernel	8×8	4×4	3×3
Stride	4	2	1
Output dimension	$119 \times 159 \times 32$	$58 \times 78 \times 64$	$56 \times 76 \times 64$
Activation function	ReLU	ReLU	ReLU

Table 3.1: An overview of the convolutional layers in the agent’s neural network.

Layer	Fully connected	Value head	Policy head
Input dimension	272384	512	512
Output dimension	512	1	7
Activation function	ReLU	Linear	Linear

Table 3.2: An overview of the fully connected layers in the agent’s neural network.

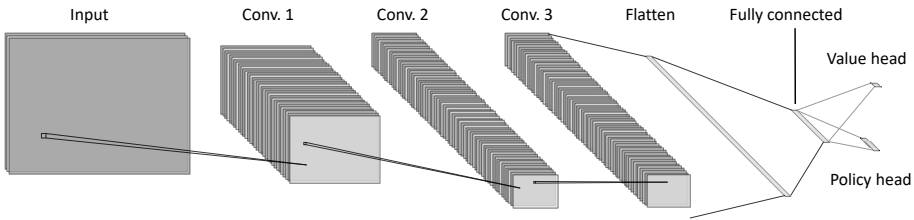


Figure 3.1: Visualization of the network described in Table 3.1 and 3.2. Base image generated by [20].

3.1.2 Environment

The environment is a 3D scene depicting a typical Swedish outdoors landscape. The scene is rendered by an engine based on OpenSceneGraph¹ that is capable of simulating visual sensors in synthetic surroundings [21]. In this environment, 10 targets are placed at random positions. In Figure 3.2 two images of the environment can be seen. The agent can interact with the environment by controlling a stationary camera’s pan, tilt, and zoom. This gives the agent seven possible actions, which are listed in Table 3.3. The pan and tilt actions are performed in discrete steps of 3.6° unless the agent is zoomed in, then pan and tilt are incremented in 0.36° steps. The agent can pan all 360° but is restricted to a downwards and upwards tilt of 45° .

¹<http://www.openscenegraph.org/>



Figure 3.2: Two images of the environment, both containing a target.

Action Nr.	Action
1	Nothing
2	Pan Left
3	Pan Right
4	Tilt Up
5	Tilt Down
6	Zoom In
7	Zoom Out

Table 3.3: A list of the agent's actions.

The observations given to the agent are images of the scene rendered from the camera's perspective. The images are RGB images with a width of 640 pixels and a height of 480 pixels. Therefore, the state given to the agent has a shape of $640 \times 480 \times 3$ where each value ranges from 0 – 255.

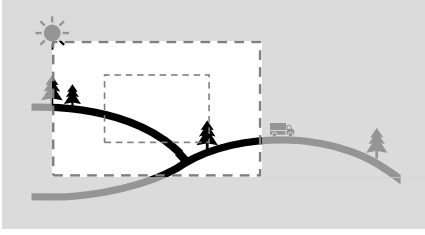
3.1.3 Resetting the Environment

The end of an episode, known as the episode being done, is defined as the agent reaching the 200th time step. When this happens, the environment is reset. This means that the agent is placed at a random position with a random rotation. The targets in the environment also get new, randomized, positions and orientations.

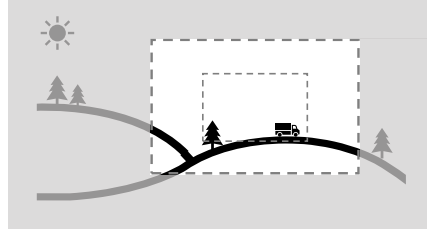
3.1.4 Goal and Reward Signal

The agent's goal is to search its environment and find a target. When a target is found, it should focus on that target by placing it at the center of its camera.

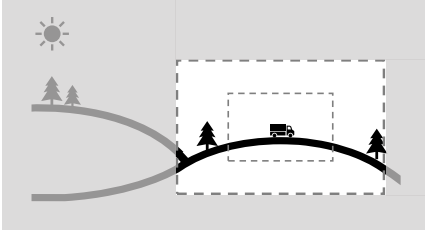
When no target is within view, the agent does not receive any reward. The agent



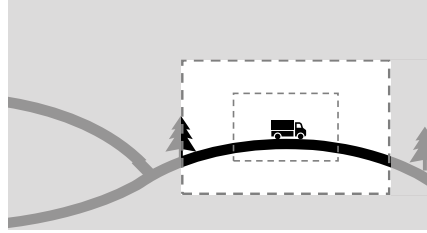
(a) No reward. Target outside of image.



(b) Smaller reward. Target within image.



(c) Larger reward. Target close to image center.



(d) Largest reward. Target close to image center and zoomed in.

Figure 3.3: The reward signal in different situations. The outer dashed square represents what the agent sees, in other words the images it is given. The inner dashed square represents the inner border which is half the distance from the image border.

only starts receiving a reward if there is a target within the image. Each time step that this is true, the agent obtains a reward relative to the distance between the target and the image center as well as the distance between the image center and half of the image border. Half of the image border is known as the inner border and it can be seen in Figure 3.3 as the inner, dashed, square. The following equation defines how the reward is calculated

$$r_{\text{dist}} = 1 - \frac{\text{Distance to center}}{\text{Distance to inner border}} \quad (3.1)$$

and as is seen, the reward decreases linearly. This means a target close to the image center gives a higher reward than a target far from the image center, see Figure 3.3. In addition to the reward based on distance, there is a reward based on how much of the screen the target takes up. A circle with a 50 pixel diameter is placed at the center of the image. This reward is the percentage of that circle's area that the target takes up in the image. The reasoning behind this reward is that the agent should have an incentive to zoom in on its target. If there are several targets within view of the agent, only the highest reward is considered.

Another reward signal that was considered was to give the agent a large reward

only when it successfully centers a target. This could, however, result in the agent rarely observing rewards since centering a target requires a long and specific sequence of actions that can be hard to find through exploration. This reward signal was not used since a reward that is too sparse can result in the agent having trouble learning [3].

3.1.5 Implementation

The agent and environment are implemented in Python. The agent is implemented with Stable Baselines 3's [22] PPO algorithm, and the environment is implemented in Gym [23] by FOI.

Following is a description of the flow of a training session which is also seen in Algorithm 3. A training session starts by initializing the agent and its weights, thereafter the environment is reset and an initial observation is obtained. A number of samples are collected by stepping the environment, also known as a trajectory. Stepping the environment is done by first letting the agent deduce an action using the latest observation. Then, based on this action, the environment communicates with the graphics engine to update it and render a new image. The reward and advantage are calculated, and the newly rendered image becomes the new observation. When a trajectory has been collected, the optimization step begins and the agent's weights are updated according to the PPO algorithm. The environment is stepped until it reaches the end of its episode, then the environment is reset whereafter training resumes. An overview of the communication between agent, environment, and graphics engine is found in Figure 3.4.

Algorithm 3: Training flow

```

Initialize agent and its weights;
while train do
    Reset environment and gather initial observation  $S$ ;
    while episode not done do
        for time step  $t = 0, 1, \dots, T$  do
            Let agent choose action  $A$  based on state  $S$ ;
            Update graphics engine according to action  $A$ ;
            Get new image (state  $S'$ ) from graphics engine;
            Calculate reward  $R$ ;
            Calculate advantage  $\hat{A}$ ;
            Check if episode done;
             $S \leftarrow S'$ ;
        end
        Update weights with PPO;
    end
end

```

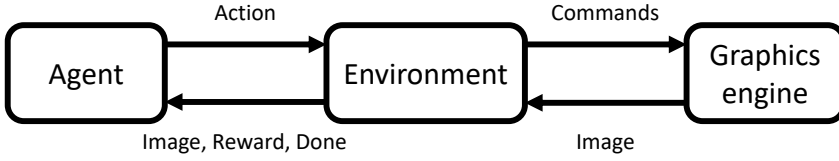


Figure 3.4: Image over the communication between agent, environment and graphics engine.

3.2 Behavior Cloning

Behavior cloning is an imitation learning method that can be used to pre-train agents. The aim of this is to give the agent a base behavior that helps it learn more efficiently when training is continued as usual. The implemented behavior cloning makes use of expert experiences which have been saved to storage. These experiences are gathered by letting a person manually control an agent while the state s_t and expert action e_t are recorded for each time step. During pre-training the loss

$$L^{\text{BC}}(a, \pi) = -\log(\Pr(\pi(s) = e)),$$

is used, as mentioned earlier in Section 2.4. As such, the agent learns to take the same action as the expert.

Due to its flexibility, another reinforcement learning library was used for behavior cloning. Instead of Stable Baselines 3, RLlib [24] was used. The implemented behavior cloning uses RLlib’s PPO algorithm but substitutes the loss function with the one seen above. It was also modified to able to read experiences from storage.

To keep the training of agents as similar as possible, it was decided that only the pre-training should be done with RLlib and that the continued training should be done with Stable Baselines 3, in accordance with the other agents. Therefore, the weights of the pre-trained agent were moved over to a Stable Baselines 3 agent before training resumed.

Two sets of pre-trained weights were created. The first set had been trained on a data set where around 200 episodes of a person manually solving the task had been collected and it was trained for 500,000 time steps. The second set of weights had been trained for 18,000 time steps on a data set consisting of 20 episodes where the expert is constantly rotating.

3.3 Performance

The performance of an agent is how well it behaves or how well it solves the given task. An accurate performance measure is hard to define, but one way of indicating an agent’s performance is through the reward it obtains. Since the

Hyperparameter	Value	Description
learning_rate (α)	0.0003	Size of gradient ascent step
n_steps	1024	Number of sampled time steps before policy update (trajectory)
batch_size	128	Number of images passed through the network simultaneously
n_epochs	10	Number of times the policy is updated using the same trajectory
gamma (γ)	0.99	Discount factor
gae_lambda (λ)	0.95	Bias vs. variance trade-off
clip_range (ϵ)	0.2	Range of clipping
vf_coef (c_1)	0.5	Value function coefficient
ent_coef (c_2)	0.0	Entropy coefficient
max_grad_norm	0.5	Clips gradient if it gets too large

Table 3.4: The hyperparameters present in the Stable Baselines 3 implementation of PPO together with a description and value. When the hyperparameter name is followed by a symbol, that hyperparameter corresponds to that coefficient in Chapter 2.3.

reward is what the agent tries to maximize, a high reward should, in most cases, mean that the agent is performing well. Therefore, when examining an agent’s performance, the reward was used to indicate of how well it performs. However, the reward most likely is not enough to solely determine an agent’s performance. Therefore, in addition to the reward and to help determine the agent’s behavior, a visual examination was performed by manually observing the agent in action.

3.4 Experiments

Four different agents were trained, and descriptions of these agents can be found in Table 3.5. All except one agent used the same hyperparameters, and these parameters can be found in Table 3.4. Agent A is the “regular” agent, and no unique changes were made to it. Agent B was the same as agent A but used an entropy coefficient of 0.01. This means that the agent had a stronger incentive to explore. Agent C and D were pre-trained with behavior cloning. Agent C used the first set of pre-trained weights where a person had solved the task manually and agent D used the second set of pre-trained weights where the expert constantly rotated.

Due to time constraints, the agents were not trained for an equal amount of time. Instead, an agent was trained for as long as its reward was deemed to go up. When the reward curve flattened out, training was stopped.

Agent	Description
A - Regular	No hyperparameters were changed and no pre-trained weights were used
B - Entropy 0.01	The entropy coefficient was changed to 0.01, promoting exploration
C - Behavior Cloning 1	Uses the first set of pre-trained weights, where the expert is a human solving the task
D - Behavior Cloning 2	Uses the second set of pre-trained weights, where the expert constantly rotated.

Table 3.5: Descriptions of all trained agents.

4

Results

This chapter presents the results gathered during training and evaluation of the agents described in Chapter 3. First, the training results are presented with plots showing how the different agents' reward changes during training. Thereafter, the behavior of the different agents is presented. Since the agent's behavior, or policy, is hard to visualize through plots and data, this section includes a description of all agents' behavior. These descriptions are based on manual observations of the agent in action. The training time of all agents was around 24 hours for every 1.2 million time steps trained, and they were trained using an RTX 2080 Ti.

4.1 Agent Training

The plots in Figure 4.1 and 4.2 show how the agents' performance changes during training. The x-axis denotes time steps and the y-axis denotes the mean reward per five episodes. Since the obtained reward is correlated to the randomized environments, the data contains large variations depending on whether the agent got hard or easy environments. The raw data containing these oscillations is plotted with a bright color in Figure 4.1. In addition to the raw data, a smoothed version is plotted with a darker color. The smoothing used is an exponential moving average which helps increase readability by suppressing high-frequency oscillations. Figure 4.2 presents the training curves of all agents in the same plot, allowing for visual comparisons.

Looking at the smooth data in Figure 4.1 we can see that agent A, B, and D rise fairly consistently until they flatten out. Agent A flattens out at around reward 40 whilst agent B and D flattens out slightly higher, at around reward 50. Agent

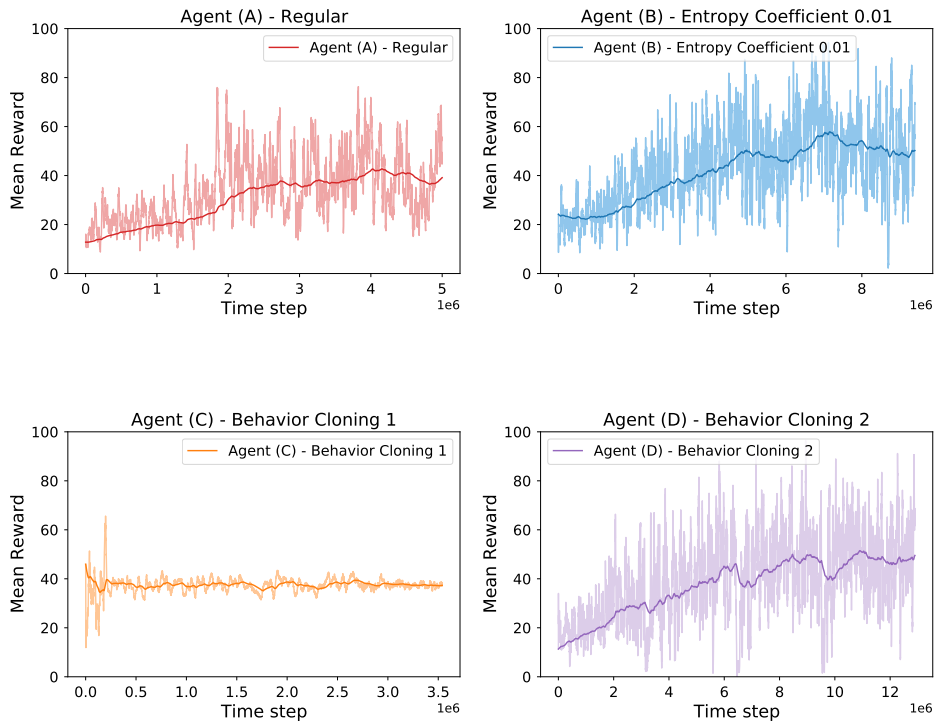


Figure 4.1: Plots showing how the agent's performance changes during training. The x-axis denotes time steps and the y-axis denotes the reward per episode. The bright line shows the raw reward and the dark line shows the smoothed reward.

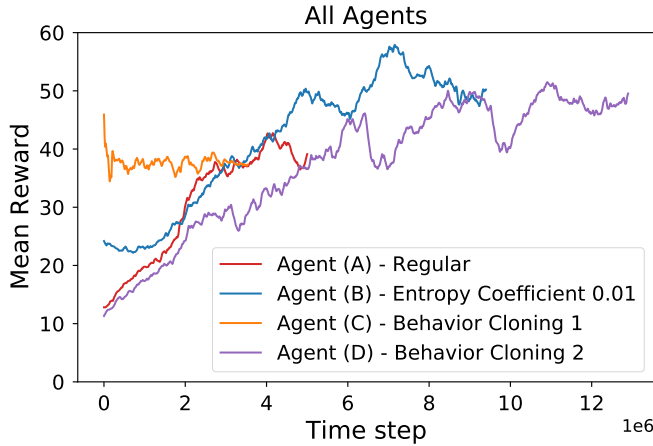


Figure 4.2: All smoothed training curves together, allowing for visual comparisons. Length differences are due to the agents being trained until their curve flattens out.

C looks different compared to the others. It starts high and then has a quick drop to slightly below the 40 reward mark after which it flattens out.

The raw reward data in Figure 4.1, shown in a brighter color, indicates that agents A, B, and D have large variations in their reward. Agent C, however, is steadier and only has slight oscillations.

Figure 4.2 shows the training curves of all agents combined into a single plot. Here we can see that agents A and C achieve roughly the same reward whilst agent B and D goes higher.

4.2 Agent Behavior

By manually examining how the agents act, three distinct behaviors can be observed. The first behavior can be seen in agent A. When no target was in sight, agent A did not show any distinct search behavior. Instead, it took what seemed like random actions. However, when a target did come into its view, the agent directed its camera to focus on the target by placing it at the center of the image. Sometimes the agent was able to keep the target centered, but other times it would rock back and forth over the target. This volatility of rocking back and forth sometimes even resulted in the agent losing sight of the target. The agent did not always succeed in detecting targets. Sometimes it would pass over them and this seemed to be related to the distance between agent and target. The further away a target was, the more trouble the agent had. In addition, the agent had not learned that there are no targets in the sky. An image of the sky did not cause the agent to tilt downwards and this sometimes led to the agent getting stuck looking up in the sky, not finding any targets.

Agent C developed a behavior of constantly rotating left. This behavior still gave it a reward that was in line with agent A.

Agents B and D can be seen as a combination of the other two behaviors. When no target was in sight, they both showed a slight search behavior of consistently panning right. They did this by choosing the action of rotating right more frequently than any other action, but as they still took other actions this search behavior was quite shaky. Sometimes it also led to the agents looking either down into the grass or up into the sky. When a target came into the agents' view they tried to place it at the center of their camera. The behavior of centering a target was similar to that of agent A and they also shared the same difficulties. Sometimes they had trouble keeping a target centered and they had trouble detecting targets far away.

Figure 4.3 shows an example of how an agent, in this case agent B, traverses its pan and tilt angles during an episode. In the episode shown the agent manages to center a target after around 75 time steps. By looking at the pan and tilt plots we see some volatility, this is the shaky behavior that was described above.

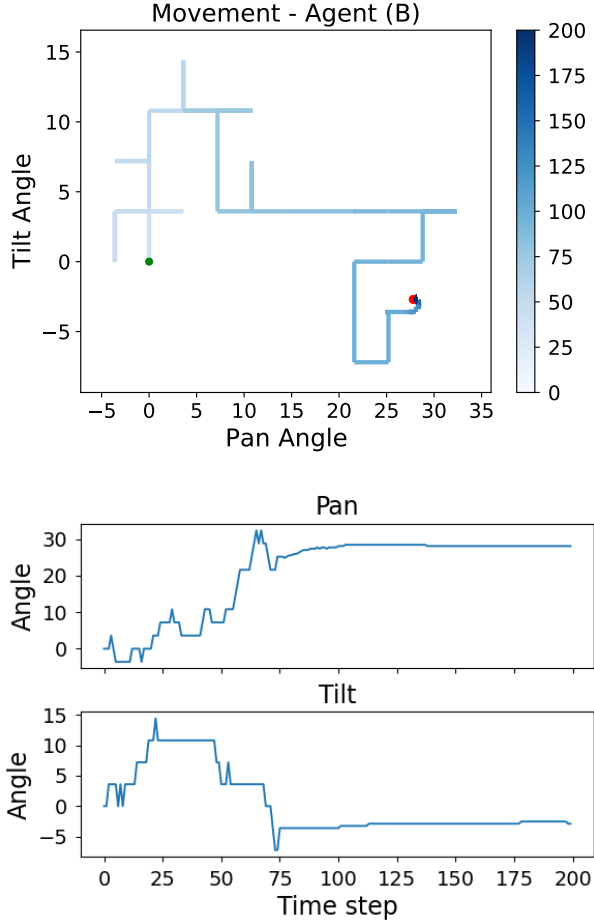


Figure 4.3: The top plot shows how agent B traverses its pan and tilt angles during an episode where it managed to find and center a target. The initial orientation of the agent is defined as angles 0 and is represented by a green dot. The red dot represents the angles the target is located at. The darkness of the line represents the number of time steps that has passed, or in other words, how many actions the agent has taken. The pan and tilt plots show how the respective angles change each time step.

5

Discussion

In this chapter, the results of the project are discussed, as well as the method used. Finally, future work is brought up.

5.1 Results

This section discusses the performance and behavior of the trained agents.

5.1.1 Behavior

Manual evaluation of agent A showed that it solved the given task to some degree, but far from perfectly. It did not appear to have learned any search behavior, but if a target was within view, it could perform the actions needed to focus on that target. The agent was able to do this with varying degrees of success. Sometimes it went well, sometimes it passed its target, and sometimes it focused on a target only to lose it after a while. The difficulty of focusing on a target seemed to be related to the distance between target and camera. The further away the target was, the less success the agent had. There is a possibility that this is related to the resolution used, 640×480 . The images contain lots of detail, and as objects get further away from the camera they lose detail as they are represented by fewer pixels. This could be a reason why the agents had trouble with targets far away, and a higher resolution may have helped.

Agent C got stuck in the behavior of constantly rotating left. It could be argued that this is a good search behavior, but since the agent never stopped, not even when passing directly over a target, it was not a desired behavior. Looking at the plots in Figure 4.1 we see that this behavior garnered approximately the same reward as agent A. When the agent rotates, it will consistently pass over targets.

Since the agent receives a reward each time step it spends near a target, this is a safe way of consistently gaining a mediocre reward. This is opposed to agent A that instead had highs of finding and successfully centering a target, and lows of not finding a target at all. These highs and lows averaged out to a reward similar to agent C. This phenomenon can be seen in the plots of Figure 4.1 where agent C only has slight variations in the raw reward due to it consistently gaining a mediocre reward. The other agents are a lot more volatile, indicating the times they did not find any target or the times they were able to quickly focus on a target.

Weird behaviors are fairly common in reinforcement learning, so an agent that constantly turned left was not surprising. Weird behaviors often come from the agent finding a local optimum or an imperfectly designed reward signal. Designing a reward signal that imposes the exact behavior we want is hard, and it is natural to settle for a simple and intuitive reward. These rewards work in many cases, but in some cases they lead to undesirable behaviors. [25] The reward signal is discussed further in Section 5.2.4 together with the performance measure.

Agents B and D achieved the highest reward and were also the agents that exhibited the best behavior. Their behavior was similar to what would be expected from an agent solving this problem. In other words, they showed a simple search pattern of rotating until they found a target, and then tried to focus on it. Their performance however left some to be desired; they were quite shaky and in some cases failed their tasks. The results are still seen as positive since if they would be able to refine their behavior, and get rid of the shakiness, they would be able to solve the task efficiently. Agent B accomplished this by having an entropy coefficient of 0.01, which is the coefficient in PPO that promotes exploration. Due to this increased exploration, we believe that the agent more frequently was able to observe cases in which it managed to rotate to find a target as opposed to agent A that did not demonstrate any search behavior. Agent D achieved this through behavior cloning, which is discussed further in Section 5.1.2.

5.1.2 Behavior Cloning

Agent D performed similarly to agent B. The initial behavior acquired from behavior cloning seems to have given the agent an advantageous exploration that allowed it to develop both a search and focusing behavior. This was, however, also managed by agent B that used an entropy coefficient of 0.01. Therefore, behavior cloning did not lead to better performance or behavior than could be achieved otherwise. However, behavior cloning still gave the agent a good initial behavior and being able to manually define such a behavior is beneficial. By using behavior cloning to induce a base behavior, we do not have to rely on random exploration to the same extent. Therefore we see potential in applying behavior cloning to similar problems that could make use of a more advanced initial behavior. Agent C also used behavior cloning but did not perform as well as agent D. A possible cause for this is that the behavior cloning was trained for too long (500,000 time steps) on a too small dataset (200 episodes) resulting in the agent

becoming overfitted.

5.2 Method

This section discusses issues of the used method.

5.2.1 Sample Efficiency

Reinforcement learning is a field that has a low sample efficiency. The number of samples, and therefore time, needed to train a well-performing agent is often high. The number of samples needed varies heavily depending on the problem and setup, but examples would be an agent learning various control tasks requiring between hundreds of thousands and a couple of million time steps [13] or an agent playing Atari games requiring tens of millions [16]. Therefore, it is valuable to have an environment that can be sampled at high speeds. In this project, the environment was quite slow as it used images rendered by a graphics engine depicting a detailed 3D scene. Rendering these images takes time, and to capture the detail of the scene an image size of 640×480 was used. Image sizes also affects training times and compared to the image sizes used in many other reinforcement learning studies [10–14] our image size was large.

One benefit of using a highly detailed environment, such as the one in this project, is that it is a better representation of reality. It does, however, increase development times and therefore also how advanced the developed agent can be. Therefore, it is important to make sure your environment suits your specific needs. If the goal is to develop an agent with an advanced search behavior, it would be better to use a simpler environment with faster sampling times and lower image sizes like in the previously mentioned environments [10–14].

5.2.2 Problem Difficulty

The problem setup in this project is a difficult one since the position and orientation of both agent and targets are randomized each time the environment is reset. Therefore, the agent cannot simply adapt to a single surrounding. It has to generalize and be able to perform independently of where in the scene it is placed. Due to the limited time of this project, it could be argued that a simpler problem setup should have been used, such as having the agent's position be constant whilst the targets' positions and orientations were randomized. That would have allowed for a better performing agent to be developed, but such an agent would most likely not have been able to generalize to different positions. Therefore, even though our results are not perfect, they may be more interesting than the results of a well-performing agent in a more static environment.

5.2.3 Neural Network

The choice of neural network may have been suboptimal since its design was quite simple. A more advanced design may have performed better and allowed the agent to more accurately detect and focus on targets, but it was decided to stick with the chosen design since trying out different designs would be too time-consuming, as mentioned in Section 1.4. A larger network would also result in even longer training times. With an inference time of 4 ms, the current network could train approximately 1.2 million time steps each 24 hours. Doubling the inference time would mean 1.2 million time steps could be trained in 38 hours. The training time is not doubled when the inference time is doubled because sampling the environment still takes the same amount of time.

5.2.4 Performance Metric and Reward Signal

One of the agents, agent C, got the undesirable behavior of constantly rotating left. One way of addressing such behavior would be to alter the reward signal. At the beginning of the project, an idea was to use a much sparser reward than is currently used. The proposed reward signal was to give the agent a larger reward only when it successfully centers a target. This would most likely have avoided the problem of a constantly spinning agent since the agent would not get any reward for simply passing by targets. The downside of this reward signal is that it is sparse; a long and specific sequence of actions is needed to center a target and this leads to the agent rarely observing any reward. It was decided against using this reward signal since rewards that are too sparse can give the agent trouble learning [3].

Another idea is to shape the reward signal so that this specific behavior is penalized. Shaping the reward to avoid specific behaviors works as long as there are only a few edge cases that have to be considered, but if there are many, shaping the reward signal becomes a hard and time-consuming task.

As mentioned in Section 5.1.2 it is hard to define a reward signal that explicitly promotes the desired behavior, and this brings to question whether the reward is a good performance metric. Is it reasonable that a constantly spinning agent gets a reward as high as an agent actually trying to solve the task by focusing on targets? And that these two are deemed equally good? In addition, it is hard to grasp how well an agent performs by simply looking at a number. To counteract this vagueness, manual observations of the agent's behavior were made and descriptions of these behaviors were added to the results.

5.2.5 Reliability and Validity

Reinforcement learning is suspect to randomness. Both in what values it is initialized with and the exploration of the environment. Exploration affects learning since it is what leads the agent to rewards, and in extension, dictates what it learns. The randomness present in reinforcement learning can therefore mean that two agents, trained with the same initial conditions, may perform differently

and these differences can be of significant size. To better understand the performance of a specific agent it would therefore have been better to train each agent several times. By running the same experiment several times, a performance measure with a confidence bound can be created, more accurately showing the agent's performance. [26] The randomness within reinforcement learning also affects the reproducibility of this work, as two agents trained with the same initial conditions do not necessarily develop the same behavior.

Further, hyperparameters were not tuned. Even though PPO is said to be a robust algorithm in the original paper [16], hyperparameters can still affect the training both in time and performance, as shown in [26]. Therefore, it would have been beneficial to perform hyperparameter tuning and make sure we got the best results possible.

As mentioned in Section 5.2.1, reinforcement learning requires lots of samples. For example, in the original PPO paper, the agent playing Atari games were trained for 40 million time steps [16]. That is a lot compared to our agents and it brings to question whether our agents had reached their top performance or if they would have improved by further training. Therefore, it would have been interesting to continue the training of our agents and see if they improve. This was however not feasible due to time constraints.

5.2.6 Source Criticism

This thesis mostly makes use of academic books and papers. A couple of the used papers only exists as preprints and are therefore not peer-reviewed. Not being peer-reviewed makes the validity and reliability of a paper poor. The papers existing only as preprints come from the OpenAI team, who mostly seem to publish their work through either preprints or blogposts. These papers were still deemed trustworthy since OpenAI are well known within the field and has an impressive track record.

As for the books, [3, 9, 15] are all famous in their respective fields, and the book [2] was deemed trustworthy as it used as course literature at LiU. [2] is however quite old, but this was not seen as a problem as only fundamental information was taken from it.

5.3 Future work

The environment in this project is partially observable. This means that the observations given to the agent do not necessarily tell the agent the exact state it is in [9]. For example, the agent is not aware of the location of a target unless it is within the agent's view. To help the agent get a better understanding of its environment, it would be interesting to investigate the use of an LSTM (Long Short Term Memory) together with extra inputs in the form of the agent's internal angles. Currently, the agent does not have access to its pan- and tilt angle. By adding this in addition to the image, and by using a LSTM, the hope is that

the agent should be able to tell what positions it has already been to. This would, hopefully, lead to the agent being able to improve its search behavior. Adding the agent's tilt angle may also help the agent learn that there are no targets in the sky and that it should look along the horizon to maximize its chances of finding a target.

It would also be interesting to investigate the use of auxiliary tasks, such as is done in [12] where an agent learns to play the deathmatch game mode in the video game Doom. The goal of the agent is to achieve as good a score as possible which includes shooting enemies. The network they use is split after the convolutional layers and this divides the network into two parts. One part is the "normal" network that has the task of learning the optimal policy, and the other part has the task of predicting whether there is an enemy on the screen. The loss from this prediction network is propagated back through the convolutional layers together with the regular loss. By training the prediction network to predict whether there is an enemy on the screen, the convolutional layers learn to extract features useful for detecting enemies. However, these features are not only useful for the prediction network, but are also useful in learning the agent's policy. [12] It would be interesting to implement this in our problem setup, to have the prediction network predict whether there is a target in the image, and see if this improves the agent's performance.

Another method that could help the agent extract relevant features is to use transfer learning. The aim of transfer learning is to transfer learned features from a pre-trained network to another. [27] Transfer learning is a common method used in image classification and it should be applicable to our problem as well. An example where this has been done is in [14], where an agent has the goal of navigating through an indoor environment using images. To do this, the agent makes use of a pre-trained ResNet-50 [28] network to extract common features found in images.

Sometimes the agent got stuck looking around in the sky. A possible way of preventing this may be to use behavior cloning. By adding demonstrations of an expert directing the agent away from the sky, the agent should be able to learn that looking in the sky is bad. The same could also be done for the case where the agent looks down into the grass. Another, more straightforward, way of solving this would be to restrict the agent from reaching these upwards and downwards angles. However, this would be less about teaching the agent good behavior and more about forcefully preventing it from making mistakes.

As mentioned in Section 5.2.1 it may be a good idea to use a simpler but faster environment. This brings up an interesting subject to investigate, of whether it is possible to speed up the rendering of our environment. Many methods exist where render times can be sped up at the cost of graphical quality. Doing this would reduce training times, but would the agent's behavior also be affected by this? And would an agent trained in an environment with low graphical quality be able to generalize to an environment with high graphical quality?

5.4 Work in a Wider Context

Autonomous systems can bring many benefits, but there are also ethical and social dilemmas coupled with them. One of the problems concerning autonomous systems is the question of decision-making, and while the system in this thesis does not have to make ethical decisions, it is possible that a system building upon this work will. The question of decision-making becomes more important as the field of artificial intelligence evolves, and this is often exemplified with autonomous vehicles. If an autonomous vehicle gets into a situation where it has to crash, whose safety should it prioritize? Should it prioritize the "driver" over a pedestrian, or vice versa? What if there are several pedestrians? And should it take other factors, such as age, into consideration? These questions have no clear answer but eventually need to be addressed. Another question is the one regarding blame. Who is to be responsible for the actions of an autonomous system? The company that sells it, the engineer that designed it, or its user? [29]

Further, this work addresses military functions as it studies how reinforcement learning can be used to create an intelligent sensor system and the task investigated is a reconnaissance task. As with all military applications there are ethical dilemmas speaking both for and against it. Some would argue that defence research makes the world a safer place, whilst others would argue the opposite.

6

Conclusions

This thesis has studied the possibilities of applying deep reinforcement learning methods to an image-based sensor control task within a detailed 3D rendered environment. The first research question asks how well an agent, using a simple neural network, performs on the given task. The best-performing agents managed to develop both a search and focusing behavior. A search and focusing behavior, in this case, means that they rotated consistently until a target was in view of the agent. They then focused on that target by placing it at the center of their camera. The performance of these agents was not perfect — their movement contained quite a bit of randomness and while there were times they succeeded in their task, there were also times where they failed. Even though the performance was not perfect, the results are seen as positive as the developed behavior has the potential of efficiently solving the task given that the behavior is refined. This indicates that this problem is solvable using methods similar to ours.

The second research question asks how the use of behavior cloning affects the behavior and performance of the agent. In this case, behavior cloning neither led to a different behavior or better performance than the other agents, but this is not to say that the use of behavior cloning was not successful. Due to the induced initial behavior, one of the behavior cloning agents managed to develop both a search and focusing behavior. This is good, but since the same behavior was achieved by an agent not using behavior cloning it did not have a significant impact. We believe that behavior cloning can be helpful in tasks similar to ours but where a more advanced initial behavior is desired. A behavior that is less likely to be stumbled upon by the agent's exploration. This thesis, however, cannot answer to what degree behavior cloning can be used to induce more advanced behaviors as these were not investigated.

Intelligent systems can bring many benefits, such as increased effectiveness, increased safety, and lowered workload. While the agents developed in this thesis are far from fully implemented, intelligent, sensor control systems, the thesis does provide grounds for future research of such systems.

Bibliography

- [1] J. E. Ormrod and K. M. Davis, *Human learning*. Merrill London, 2004.
- [2] S. Marsland, *Machine Learning: An Algorithmic Perspective*. CRC press, 2009.
- [3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, second ed., 2018.
- [4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [5] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, “Mastering the game of go without human knowledge,” *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [6] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, “Dota 2 with Large Scale Deep Reinforcement Learning,” *arXiv preprint arXiv:1912.06680*, 2019.
- [7] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [8] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, “Solving Rubik’s Cube with a Robot Hand,” *arXiv preprint arXiv:1910.07113*, 2019.
- [9] S. J. Russell and P. Norvig, *Artificial Intelligence : A Modern Approach*. Pearson series in artificial intelligence, Pearson, 2021.
- [10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level

- control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [11] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “Vizdoom: A Doom-based AI Research Platform for Visual Reinforcement Learning,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8, IEEE, 2016.
 - [12] G. Lample and D. S. Chaplot, “Playing FPS games with deep reinforcement learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.
 - [13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” in *International Conference on Learning Representations (ICLR)*, 2016.
 - [14] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning,” in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3357–3364, IEEE, 2017.
 - [15] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, vol. 1. MIT press Cambridge, 2016.
 - [16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
 - [17] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust Region Policy Optimization,” in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.
 - [18] C. E. Shannon, “A Mathematical Theory of Communication,” *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
 - [19] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations,” in *Proceedings of Robotics: Science and Systems XIV*, 2018.
 - [20] A. LeNail, “NN-SVG: Publication-Ready Neural Network Architecture Schematics,” *Journal of Open Source Software*, vol. 4, no. 33, p. 747, 2019.
 - [21] F. Näsström, J. Allvar, J. Berggren, D. Bergström, V. Deleskog, R. Forsgren, J. Hedström, and P. Lif, “Simulering av sensorsystem för sensorvärdering,” FOI-R-4283-SE, 2016.
 - [22] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, “Stable Baselines3.” <https://github.com/DLR-RM/stable-baselines3>, 2019.

- [23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.
- [24] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica, "RLlib: Abstractions for Distributed Reinforcement Learning," in *International Conference on Machine Learning*, pp. 3053–3062, PMLR, 2018.
- [25] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," *arXiv preprint arXiv:1606.06565*, 2016.
- [26] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning that Matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.
- [27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in Neural Information Processing Systems*, vol. 27, 2014.
- [28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- [29] J.-F. Bonnefon, A. Shariff, and I. Rahwan, "The social dilemma of autonomous vehicles," *Science*, vol. 352, no. 6293, pp. 1573–1576, 2016.