ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester Thesis

Reinforcement Learning for Implicit Landing Site Detection and Autonomous Landing of MAVs

Autumn Term 2020

Supervised by:
Michael Pantic
Tim Kazik

Author:
Nasib Adriano Naimi

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

REINFORCEMENT LEARNING FOR IMPLICIT LANDING SITE DETECTION AND AUTONOMOUS LANDING OF MAVs

Authored by (in block letters):

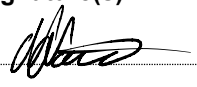
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
NAIMI	NASIR ADRIANO

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Singapore, 19.03.2021	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

Abstract

This project aims to show the feasibility of using reinforcement learning for the task of landing autonomous MAVs in unknown environments. In this work, a framework is presented for training and testing reinforcement learning agents in photorealistic simulation environments built on Unreal Engine. This framework is then used to develop a proof concept in a simple environment using a simple agent. The policy are trained with PPO using different reward shapes. The effects of reward shaping on the learning outcome of the agent are explored followed by an evaluation and discussion the respective reward configurations.

Symbols

Symbols

ψ	yaw angle
r_t	reward at time t
a_t	action at time t
s_t	state/observation at time t
θ	agent model parameters θ
π_θ	policy defined by parameters θ
A_t	advantage function
$\hat{A}_t$	advantage function estimator
$\hat{g}$	policy gradient estimator
γ	discount factor
λ	bias-variance tradeoff factor
c_{FOV}	scaling factor for field of view reward
c_{Height}	scaling factor for height penalty
c_{Step}	scaling factor for step penalty

Acronyms and Abbreviations

RL	Reinforcement Learning
VLOS	Visual Line of Sight
EVLOS	Extended Visual Line of Sight
UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
LSD	Landing Site Detection
REMODE	REgularized MOnocular Depth Estimation
SSL	Self-Supervised Learning
ROI	Regions of Interest
PG	Policy Gradient
VPO	Vanilla Policy Optimization
TRPO	Trust Region Policy Optimization
PPO	Proximal Policy Optimization
GAE	Generalized Advantage Estimator
A2C	Actor Critic
SGD	Stochastic Gradient Descent
FOV	Field of View

Chapter 1

Introduction

Autonomous landing is and will be essential to the function and mass deployment of autonomously operated micro aerial vehicles (MAVs). Tasks ranging from drone delivery to surveillance require that MAVs be able to land safely in the event of a malfunction or to approach a desired target. Especially in situations where an operator is overseeing the flight of a MAV which is beyond the visual line of sight (VLOS) and flying over treacherous terrain such as dense urban areas, glaciers, or similarly course terrains, being able to land safely in the event of a system malfunction or critically low battery proves to be challenging. In such situations, having a reliable onboard autonomous landing algorithm can increase the safety with which MAVs are operated and increase the range of situations in which MAVs can be deployed without posing a risk to bystanders or the hardware.

1.1 Problem Statement

The problem statement was formulated as follows

> Can reinforcement learning be used to land a MAV autonomously and safely in situations where an emergency landing is required?

The necessary assumptions made for this problem statement can be summarised in

1. The time for landing the MAV is limited
2. The MAV is equipped with a downward facing camera
3. The environment in which the MAV needs to land in is unknown (prior free)

1.2 Related Works

To the best of our knowledge, no work has been done that specifically attempts to use reinforcement learning for landing MAVs autonomously. In this section we will present some research and related works on autonomous landing of MAVs to give the reader a sense of what approaches have been investigated thus far. The papers mentioned illustrate different approaches for detecting suitable landing sites in unknown environments with the help of monocular vision alone.

The first approach relies on elevation mapping presented by Forster et al. In [1], they propose an on-board, single camera elevation mapping system that runs in real-time and continuously. The approach relies on a modified version of the *REMODE*

(REgularized MOnocular Depth Estimation) algorithm to compute a probabilistic 2D robot-centric elevation map. The coarse elevation map allows for a lower resolution which is deemed as sufficient for performing autonomous maneuvers. These elevation maps can then be used for landing site detection (LSD) by taking 3D points $\mathbf{p}$ located on the terrain and computing the squared difference between the elevation of point $\mathbf{p}$ and all points lying inside of the radius r. This represents the cost function and a threshold for the cost function can be specified to define a safe landing site. The resulting grid of bits then undergoes a distance transform and the the point furthest from unlandable area that is classified as safe is chosen as the landing location.

Another approach presented by Ho et al. uses self-supervised learning (SSL) based on optical flow to detect obstacles that may be obstructing a landing site [2]. Here, a MAV is trained over the course of its flight to detect, using optical flow, which points collectively represent an obstacle by measuring if they move faster than the background. That area is then selected as an obstacle and the model is trained on that sample. Once a landing is required, the network infers, using the downward facing camera image as input, where the obstacles lie and plans it's trajectory accordingly.

The final method we would like to highlight, is a segmentation based approach for fixed wing unmanned aerial vehicles (UAVs) presented by Hinzmann et al in [3]. Here, a RGB image produced by an onboard, downward facing camera is used to produce a segmented image. The segments are filtered down to the largest, connected areas and each is then classified as either potentially safe surface or not. The regions of interest (ROI) are further analyzed for their color and texture. Throughout the landing approach, the regions are tracked and a decision layer produces a mask designating suitable landing sites, taking into account the terrain roughness and slope.

1.3 System Overview

In this project, we aim to provide an alternative method for detecting suitable landing sites without explicitly classifying areas in an image. Instead, we use reinforcement learning (RL) to train a policy that generates actions to steer a MAV to a safe landing site. We present a pipeline for training the agent in a simulation built on Unreal Engine [4] utilizing the unrealCV plugin [5] and extending the interface with OpenAI Gym [6] that was created by Zhong et al. [7]. Further, the feasibility of the approach is demonstrated through a simple proof of concept and the effects of reward shaping on the end behaviour are assessed. In diagram 1.1 the System Block Diagram is depicted which illustrates the various modules and interfaces used in the pipeline.

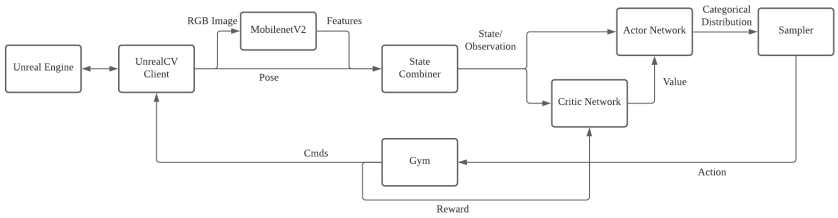


Figure 1.1: Block Diagram of System Modules

Chapter 2

Reinforcement Learning: Framework and Optimization

2.1 Introduction to Reinforcement Learning

Reinforcement learning represents a paradigm of machine learning with two principal components; an agent and an environment. An agent is tasked with learning how to act in an uncertain environment in order to maximise a numerical reward signal [8]. At each step, an agent samples an action a_t from a conditional probability distribution π_θ conditioned on the state s_t. This action in turn generates feedback from the environment in the form of a reward r_{t+1} and the new state s_{t+1}. The goal of the agent is to maximize the cumulative reward through it's actions, hence the objective can be formulated in a way which permits the parameters Θ of the policy to be learned through gradient ascent, similar to supervised learning problems. Here, we focus on on-policy, model-free based methods.

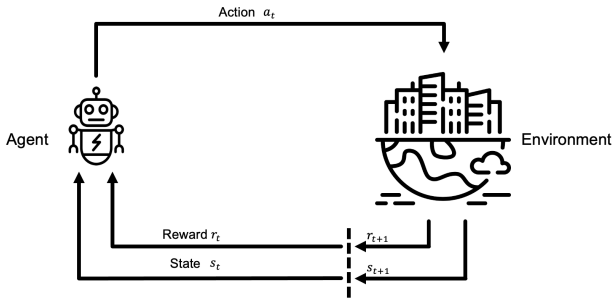


Figure 2.1: Illustration of the basic interaction between an agent that is placed in an environment to learn.

Unlike other domains in machine learning, reinforcement learning encounters the challenge of exploitation vs. exploration [8]. On the one hand, the agent needs to perform the actions that led to high rewards in the past to keep obtaining similarly high rewards. On the other hand, it needs to discover the actions that lead to higher rewards on the long term, meaning it must try actions it has not performed before. This trade-off represents the dilemma of reinforcement learning for finding an optimal policy, since neither of the two can be pursued exclusively.

For policy gradient (PG) methods to work, an estimate of the policy gradient must be computed [9]. The gradient estimate can then be used in stochastic gradient as-

cent to learn the policy. The most commonly used and simplest form of the gradient estimator that is referenced as the vanilla policy optimization (VPO) is formulated as

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t(s_t, a_t)]. \tag{2.1}$$

which results from the objective function

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t(s_t, a_t)] \tag{2.2}$$

where π_θ is a stochastic policy, $\hat{A}_t$ is the advantage function which estimates the relative value of the selected action, and for an algorithm that samples and optimizes alternatively $\mathbb{E}_t[...]$ represents the empirical mean of a finite set of samples. Note that performing multiple optimization steps on the same trajectory empirically often leads to large policy updates that are unfavourable [9]. This motivates alternative policy optimization methods, most notably the proximal policy optimization.

2.2 Proximal Policy Optimization

Training policies with VPO leads to sub-optimal policies, since even in the case that parameters are kept close to each other in parameter space between updates, these small changes can result in large changes in the probability space. This means that the risk of converging at sub-optimal policies is higher and using large step sizes may be dangerous, leading to the collapse of the policies performance. Trust region policy optimization (TRPO) [10] maximizes a *surrogate* objective function

$$\underset{\theta}{maximize}\, \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t(s_t, a_t)] \tag{2.3}$$

with the policy update being constrained by a KL divergence constraint

$$\hat{\mathbb{E}}_t[KL[\pi_{old}(\dot{|}s_t), \pi_\theta(\dot{|}s_t)]] \leq \delta. \tag{2.4}$$

In practice, TRPO makes use of conjugate gradients to approximate a solution and a computational discount. However, conjugate gradients are complicated to implement and provide limited flexibility. The objective can be reformulated using $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$, meaning that $r(\theta_{old}) = 1$. TRPO then maximizes the *surrogate* objective where CPI stands for conservative policy iteration [9]. The objective is then defined as

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}\hat{A}_t(s_t, a_t)] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t(s_t, a_t)]. \tag{2.5}$$

Note that without constraining the maximization of this objective, a policy could be updated in excessively large steps. This would lead to policy collapse.

A simpler method, proximal policy optimization (PPO), was introduced by Schulman et al. in [9]. PPO methods provide the same stability and reliability as TRPO methods, but allow for more flexibility in choice of architecture and have better overall performance [9]. This method formulates the main surrogate objective as

$$L^{CLIP}(\theta) = \min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right). \tag{2.6}$$

Note that ϵ is typically chosen as $\epsilon = 0.2$ but can be chosen as a tunable hyper parameter. This objective contains the CPI objective as the first term of the min function and the second term is the clipped CPI objective. The modified surrogate

in the second term clips the probability ratio in order to remove the incentive for moving the probability ratio outside of the interval $[1-\epsilon, 1+\epsilon]$, that prevents the optimization from taking steps in the policy update that significantly change the probability space [9]. The final objective is the minimum of the clipped and unclipped surrogate objective, meaning that the modified objective represents a lower bound of the unclipped objective. Thus, the probability ratio is ignored if it improves the objective and included if the objective is worsened. Note that PPO also makes it easier to train agents with neural networks that share parameters between value function and policy.

The framework of the agent chosen for the agent is the Actor Critic (A2C). In this setup, there is a network that yields the conditional probability distribution and a separate network that criticises the action. In this case the critic is represented by the value function estimate that takes the current state as input to derive the value of the rest of the episode. Thus the complete objective is formulated as

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{2.7}$$

where $L_t^{VF}(\theta)$ denotes the squared-error loss of the value function

$$L_t^{VF}(\theta) = (V_\theta(s_t) - V_t^{targ})^2 \tag{2.8}$$

and $S[\pi_\theta](s_t)$ represents an entropy bonus that encourages exploration.

2.3 Generalized Advantage Estimation

PPO makes use of the advantage function estimators when computing the policy gradients to reduce the variance of the policy gradient. The advantage function is defined as the relative value of the selected action compared to the average action and is defined as

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{2.9}$$

where $A^\pi(s_t, a_t)$ is the real advantage function, $Q^\pi(s_t, a_t)$ is the Q-value function for a state-action pair, and $V(s_t)$ is the value function. This can be interpreted as Q being the return from time t till the end of the episode or time horizon and the value function being the baseline estimate of the expected return. The purpose of the advantage function is to reduce the variance of the gradient estimates, that can otherwise lead to unreliable and very skewed gradient estimates. Especially for longer time horizons, the variance of the gradient estimator scales to intolerable levels [11]. For policy methods, estimates of the advantage function are used to calculate the policy gradient. The choice of advantage function estimator allows us to reduce the variance of the gradient estimator further.

The advantage function can be estimated by a k-step estimate of the returns and subtracting the baseline $-V(s_t)$

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V \tag{2.10}$$

where γ is the discount factor and δ_t^V

$$\delta_t^V = -V(s_t) + r_t + \gamma V(s_{t+1}). \tag{2.11}$$

For the infinite time horizon this yields

$$\hat{A}_t^{(\infty)} = \sum_{l=0}^{\infty} (\gamma^l \delta_{t+l}^V = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{(t+l)} \tag{2.12}$$

which is exactly the empirical returns minus the baseline [11].
The generalized advantage estimator $GAE(\gamma, \lambda)$ is then defined by Schulman et al. as

$$\hat{A}_t^{GAE} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \tag{2.13}$$

which is the exponentially weighted average of the k-step estimators as stated in [11].

The generalized advantage estimator (GAE) introduces an extra hyperparameter $\lambda \in [0, 1]$ that adjusts the trade-off between variance and bias [11]. The training and sampling setup used first samples T steps and then updates the policy on those T steps before sampling anew. This setup requires that the GAE be truncated such that it does not look beyond the T timesteps.

2.4 Choice of Optimization and Networks

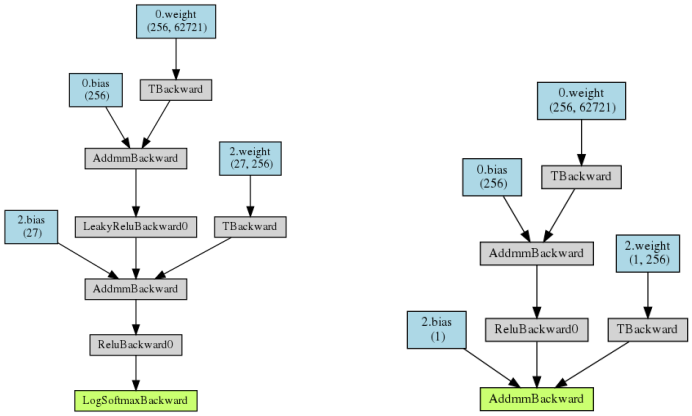


Figure 2.2: Left shows the actor network and right shows the critic network.

In this project, PPO was used together with GAE for a discrete action space. Both the actor and critic network were kept small, each with 2 layers and 256 hidden neurons each. The actor network outputs the log softmax of the different actions available in the action space. These represent the probabilities of each action in a categorical distribution. Furthermore, Log softmax and leaky ReLU were used instead of softmax and ReLU to improve the numerical stability during backward propagation.

Chapter 3

Simulation Environment and Agent Design

In order to train a reinforcement learning agent, a simulated environment is required. Depending on the problem setting, a simulation needs to be more or less elaborate. For this problem statement with the assumptions made, photorealistic graphics similar to those seen in many games today were required.

3.1 Evaluation of Simulation Environments

As stated in section 1.1, the MAV that poses as our agent has a downward facing camera with an image plane parallel to the floor beneath it. The agent makes use solely of the image it receives and pose information. In order to train an agent properly in this setting, the simulation must produce camera images with similar fidelity and resolution as would be seen on a real on-board camera. This sets the requirement of a photorealistic simulation. Furthermore, the agent needs to detect a collision with solid objects or the surface below. Thus, it must simulate the physics of the surrounding objects as well as it's own. In an effort to standardize the environments used for reinforcement learning and offer benchmarks for comparison of algorithms on different problems, Open AI proposes an interface for environments that is open source called Gym. Hence, the final prerequisite for the simulation environment was that it can interface with Open AI's Gym.

Criteria	Flightmare	Airsim	UnrealEngine
Photorealistic renderings	✓	✓	✓
Simulates Physics and Collisions	✗	✓	✓
Compatible with OpenAI Gym Interface	✓	?	✓

Table 3.1: Comparison of different simulation environments.

The three criteria above were used to assess the feasibility of different simulation environments. The first environment evaluated was Flightmare developed by Song et al. [12]. Flighmare was designed to have disconnected and individually configurable rendering engine built on Unity [13] and physics simulation for dynamics. However, collisions with the scene were not simulated by the physics engine, meaning that the simulated MAV was able to fly through walls etc. For this reason, Flightmare was not suitable as a simulation environment.

Another option evaluated was Airim [14], Microsoft's open source simulator for drones, cars and more built on Unreal Engine. At the time of evaluation, this simulation environment seemed too complex for the initial proof of concept in mind and the compatibility with Open AI's Gym was still unclear. Since the completion of the project, a wrapper has been implemented for Open AI's Gym. Nonetheless, at the time of evaluation of the simulation environments, AirSim did not provide the interface for Open AI's Gym and was deemed to complex for the first proof of concept.

The last option evaluated was the option of using Unreal Engine and Unreal Editor to construct a custom scene and pawn with the unrealCV plugin [5] that is used to construct the interface with Open AI's Gym through the extension of the interface Gym-unrealCV [7] implemented by Zhong et al. This setup satisfied all our criteria without sacrificing flexibility but required extending the interface between Gym and the simulation.

3.2 UnrealEngine, UnrealCV, and Interface with OpenAI Gym

The scene and pawn of the simulation were constructed on the Unreal Engine using the Unreal Editor. The packaged game included the unrealCV plugin which provides a server in the game instance which can be queried by a client to get RGB images from the cameras embedded in the pawn, the pose of the pawn and other information such as a segmented image. The unrealCV client also provides the interface for manipulating the pawn in the scene. Gym-UnrealCV provides the necessary functions for initialising a unrealCV client and for sending requests through the client. These represent the interfacing modules that are used to define the Gym environment. The custom Gym environment extends the base Gym environment class and defines the action and observation space and overwrites the functions step and reset. For more information on the gym interface, we ask the reader to consult the relevant documentation.

3.3 Simulation Scene

The scene used for training and testing the agent was designed to provide a simple proof of concept. In this scene the green surface is considered safe landing location whereas the red an unsafe landing location. The checkerboard surface is enclosed by invisible walls to prevent the agent from leaving the scene.

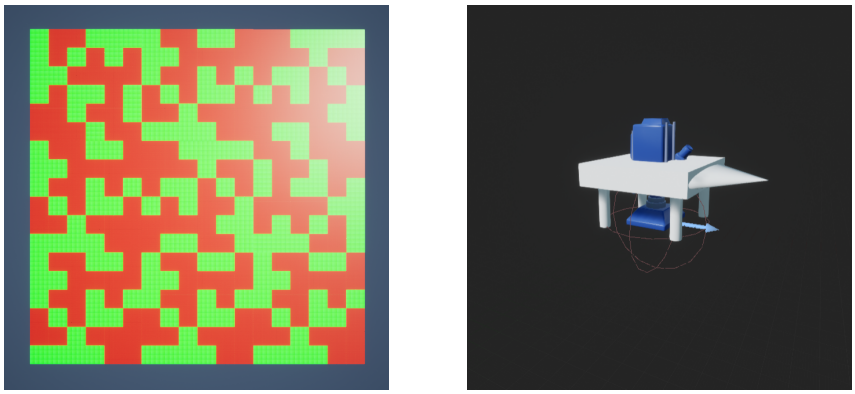


Figure 3.1: Left shows the checkerboard scene and right shows the simple agent.

3.4 Agent Design

The agent was designed with simple movements in mind. In fig. 3.1 the pawn which represents the agent in the simulation is shown. The pawn has a downward-facing camera and can move along all three axis as well as being able to yaw. It cannot roll or pitch, which is in line with the the behaviour expected for an emergency landing since a MAV would not perform aggressive maneuvers in that would cause it to roll or pitch noticeably. The pawn is what allows the agent to move in the simulated environment and the agent also needs to interface with Gym. In order to interface with gym a observation and action space need to be defined.

The observation represents the state and consists of features of the RGB image produced by the camera on the pawn and the z coordinate of the pose. The features of the image are provided by a feature extractor which in this case was chosen to be a pretrained version of the Mobilenet V2 model with the classification layers removed. In chapter 5 the observation space was expanded for some experiments to contain further information.

The action space represents the all possible actions that can be taken in single step by the agent. It can be either discrete or continuous. Here, it was chosen as a discrete set of 27 actions, where each action represents a movement to one of the vertices depicted in fig. 3.2. Note that none of the actions changed yaw.

After concluding an episode, the agent needs to be reset. The reset was defined as a uniform sampling of x,y,z coordinates in the confines of the scene as well as uniformly sampled yaw from the interval $[0, 2\pi]$.

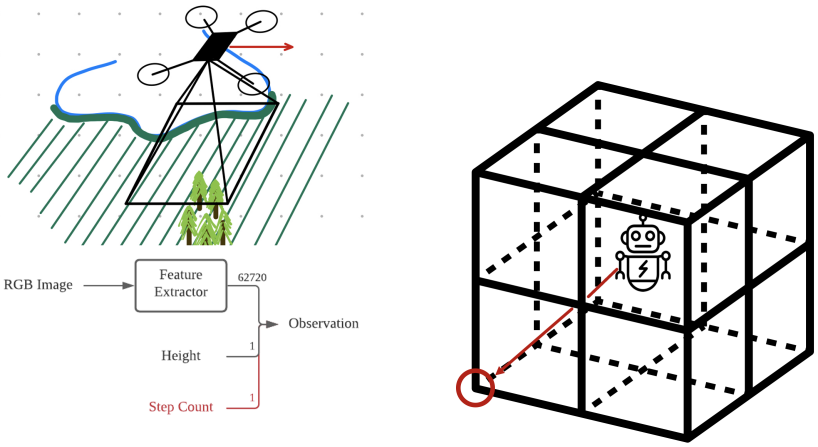


Figure 3.2: Left shows a graphic of the observation space as well as block diagram of the combination of the observations, right shows the action space.

Chapter 4

Reward and Reward Shaping

In reinforcement learning the agent is tasked with achieving a goal or set of goals. These goals are formalized by means of a reward signal that is passed from the environment to the agent [8]. A reward is distributed as a single scalar number at each timestep and the goal is to maximize the cumulative reward for the entire episode. This can be reformulated as the *reward hypothesis*: Goals and purposes can be thought of the maximization of the expected value of the cumulative sum of reward received [8].

4.1 Introduction Reward Shaping

Reward shaping, first introduced in [15], describes the process of defining a reward signal that ultimately defines how the goal is interpreted by the agent, thus affects the overall performance of the agent. Incorrectly formulated reward signals can lead to what is known as the *cobra effect*, which occurs when incentive structures have an adverse effect on the problem and exacerbate it rather than solving it. Hence why reward shaping is the key challenge in a reinforcement learning problem; finding a way of incentivizing the right behaviour. Often times a problem may be binary; either the landing was successful or it was not. In these cases the most basic reward formulation is to distribute a reward for a successful landing or charge a penalty for a bad landing. These reward formulations are known as sparse rewards. Sparse rewards provide little feedback to the agent throughout the episode, other than the final reward signal. As a result an agent learns very slowly in this scheme due to the limited feedback. Reward shaping attempts to supplement these sparse rewards through continuous reward signals that provide feedback throughout the episode.

4.2 Objectives of Reward Function

For the problem of autonomous landing, we define a successful landing as the agent touching down on a suitable landing spot in a finite number of steps. The landing can be dissected into three concrete goals:

1. Reducing altitude and approaching the surface
2. Finding a suitable landing spot
3. Taking the shortest path to a successful landing

To achieve these three goals, the reward signal was composed of different terms catering to one of three goals each. In this section we introduce the various terms

as well as the expected effect of including these in the final reward. The evaluation of the effects of the different reward terms are discussed in chapter 5.

4.2.1 Field of View

The agent needs to detect suitable landing sites implicitly and navigate towards these. This means that the agent must be rewarded if it approaches suitable landing areas or improves it's field of view (FoV). To encourage the agent to navigate to location where it sees more suitable landing sites, a reward based on the field of view is formulated. The reward is distributed according to the fraction of the onboard camera's image that is classified as landable, $l \in [0, 1]$, through a non-linear function defined as

$$R_{FOV}(l) = (tanh(\frac{1}{9}(2\pi l - \pi)) + tanh(\frac{1}{2}(2\pi l - 1.5\pi))) \tag{4.1}$$

4.2.2 Height

To prevent an agent from solely hovering over suitable landing sites, a height penalty is introduced. The height penalty distributes a penalty according to the normalized height h by means of the non-linear function

$$R_{Height}(h) = tanh(h) \tag{4.2}$$

4.2.3 Termination Reward

The sparse reward at the end of the episode is defined as a step reward or penalty for a successful or failed landing. We denote this reward as $hasLanded$.

4.2.4 Step Penalty

A step penalty is introduced to encourage the agent to take the shortest paths to the closest landable area. It was defined as a linear function, where the step count is denoted as s. The step penalty was formulated as

$$R_{Step}(s) = -s \tag{4.3}$$

4.3 Final Reward Function

Multiple configurations of these different reward terms were tested and evaluated. A discussion of the notable experiments and their outcome follows in chapter 5. The final reward that produced the best policy was defined as

$$R(l, h, s) = 100R_{FOV}(l) + 300R_{Height}(h) + hasLanded \tag{4.4}$$

Chapter 5

Experiments

5.1 Training Setup

To train the different policies, a Nvidia Titan RTX was used and for training on 15000 samples and validating on 50 episodes, a compute time of on average 8 hours was measured.
The policies were trained online with a fixed max number of steps or samples. For each policy update, a prespecified number of steps T were sampled and the policy was updated using minibatch SGD. Note that T steps are far fewer than the max step count allowed for an episode. The number of samples T per policy update is an additional hyerparameter that can be tuned. A full set of hyperparameters used for training the policies evaluated below can be found in the appendix.

5.2 Comparison of Different Reward Configurations

All reward configurations contain the step reward issued at the end of an episode. Here we compare a few configurations of the different reward terms scaled by factors and interpret their effects.

5.2.1 Using Field of View and Height Reward Terms

This configuration consisted of the field of view reward term and the height reward term. Both terms were scaled by a separate factors. For fig. 5.1 the complete reward function was defined as

$$R(x, z) = c_{FOV} R_{FOV}(x) + c_{Height} R_{Height}(z) + hasLanded \tag{5.1}$$

$$c_{FOV} = c_{Height} = 100. \tag{5.2}$$

The reward surface approaches an almost flat configuration which leads to poor updates of the policy gradient. The result is a policy which remains almost always in the same plane and samples very few different actions. This can be attributed to the fact that the penalty for not decreasing the height is not weighted strongly and the agent opts for actions that may improve the FOV slightly. The lack of downward pressure on the agent prevents it from exploring actions which move it downward, further exacerbating the problem.

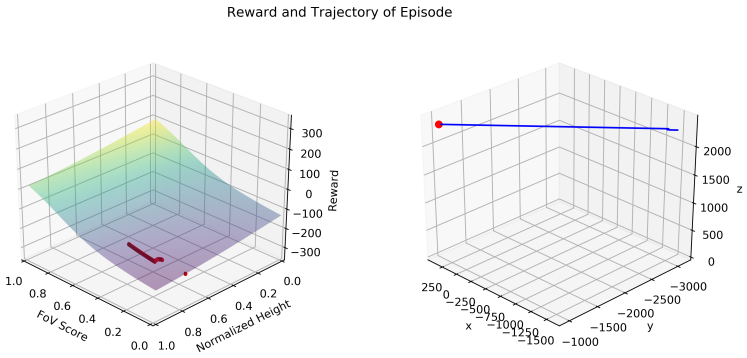


Figure 5.1: Left: The reward surface spanned by the FOV and height with the collected rewards shows as red spheres on the surface. Right: the trajectory taken by the the agent to land successfully. In this episode, we see that the reward stays in a similar region with the agent drifting in a plane parallel to the surface without making any attempts of approaching the surface.

In fig. 5.2 the reward signals for ten different episodes show a almost flat remaining reward with a flat slope across the episode length. In this configuration the agent learned primarily to move across the board and the episodes terminate in a collision with the bounding wall of the surface. The downward spikes in reward signal are due to the hasLanded reward term which issues a penalty upon collision and terminates the episode.

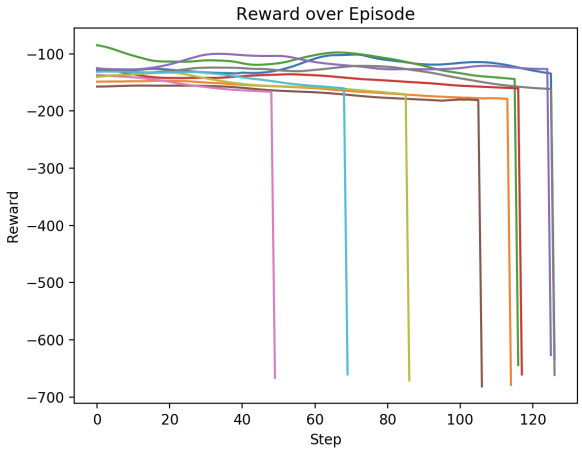


Figure 5.2: The reward signals of ten different episode for an agent using a policy trained in the above stated configuration. The reward signals remain flat and terminate in a collision.

To amend the low pressure posed on the reducing height, the factors of the above stated reward function were altered to

$$R(l,h) = c_{FOV}R_{FOV}(l) + c_{Height}R_{Height}(h) + hasLanded \tag{5.3}$$

$$c_{FOV} = 100, c_{Height} = 300. \tag{5.4}$$

This yields a more aggressive slope in the reward surface that should encourage the agent to move downward more quickly without relieving the incentive posed by improving the FOV. A successful episode can be seen below in fig. 5.3.

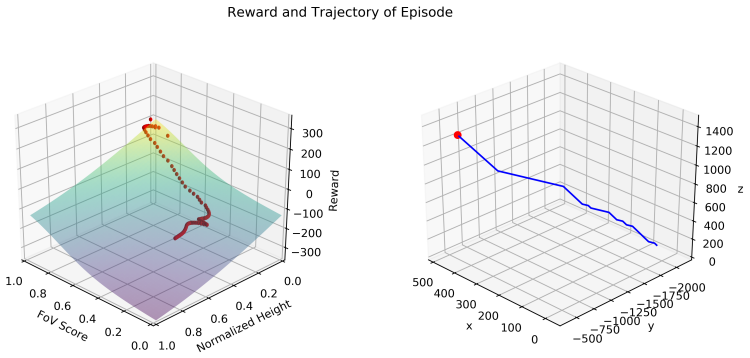


Figure 5.3: Left: The reward surface spanned by the FOV and height with the collected rewards shown as red spheres on the surface. Right: the trajectory taken by the the agent to land successfully. We see that the the reward terminates in the upper most corner of the reward surface, where the FOV is almost entirely composed of landable area and the height reaches zero.

The reward signals of ten separate episodes shown in fig. 5.4 are a lot more animated than in the previous configuration. It may be that the increase in weighting of the height reward encourages the agent to explore more downward action that may incur an immediate reduction in reward due to the deterioration of the FOV reward, but improve the cumulative reward long term. Here we see fewer episodes terminating in a collision and more intelligent behaviour overall.

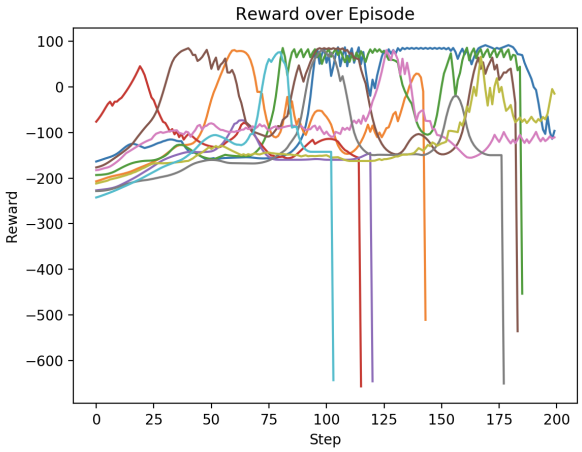


Figure 5.4: The reward signals of ten different episode for an agent using a policy trained in the above stated configuration. The reward fluctuates more starkly as it progresses through the episode and fewer episodes end in a collision.

5.2.2 Using FoV Reward, Height Penalty, and Step Penalty

The policy that was generated by the reward composed of the FOV and Height produced functional behaviour, however it did not create a policy which attempts to land as quickly as possible, even when trained with early termination as shown in section 5.3. In order to encourage more aggressive landing behaviour, a step penalty was introduced as an additional term in the reward function and the policy was trained with early termination

$$R(l,h,s) = c_{FOV}R_{FOV}(l) + c_{Height}R_{Height}(h) + hasLanded + c_{step}R_{Step}(s) \quad (5.5)$$

$$c_{FOV} = 100, c_{Height} = 300, c_{step} = 10 \quad (5.6)$$

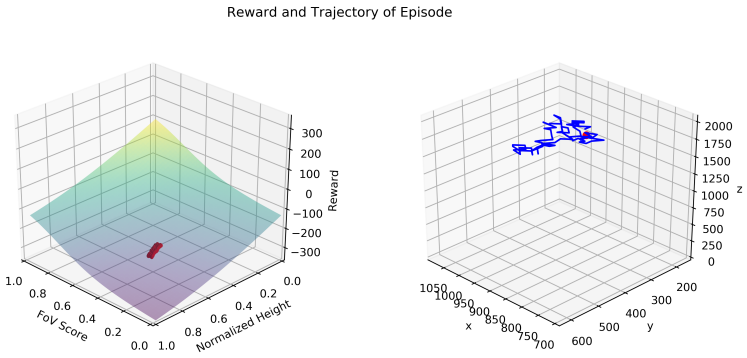


Figure 5.5: Left: The reward surface spanned by the FOV and height with the collected rewards shown as red spheres on the surface. Right: the trajectory taken by the the agent to land successfully. The agent seems confused as to which way it can go to increase it reward since all directions reduce its reward due to the step penalty. The result is random walk like behaviour.

The hope in this was that the agent would be encouraged to take actions that would terminate the episode rapidly but the results were the contrary. Introducing the step penalty resulted in random walk like behaviour of the agent as can be seen in fig. 5.5. Assessing the reward signals produced in 10 different episodes shown in fig. 5.6 there is no clear upward slope throughout the reward signals and the reward signals remain relatively flat.

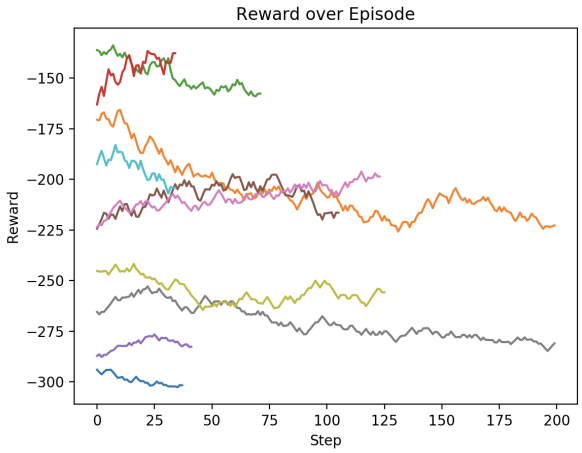


Figure 5.6: The reward signals of ten different episode for an agent using a policy trained in the above stated configuration. Rewards remain flat and do not converge at any step. General flat slope, even slightly downward slope throughout episode's reward signals.

5.3 Effects of Early Termination in Training

The policy produced by the reward scheme mentioned in section 5.2.1 produced promising but not yet satisfying behaviour. Lack of exploration during the training process due to stationary behaviour was suspected so early termination in the training setup was introduced. Early termination terminates the episode being sampled during training if the agent samples the action of staying stationary too often. The reward configuration used for training this policy with early termination was

$$R(l,h) = c_{FOV}R_{FOV}(l) + c_{Height}R_{Height}(h) + hasLanded \tag{5.7}$$

$$c_{FOV} = 100, c_{Height} = 300. \tag{5.8}$$

The result of introducing early termination was an overall better performing policy which can be seen in fig. 5.10. fig. 5.7 shows one of the successful episodes produced by this policy.

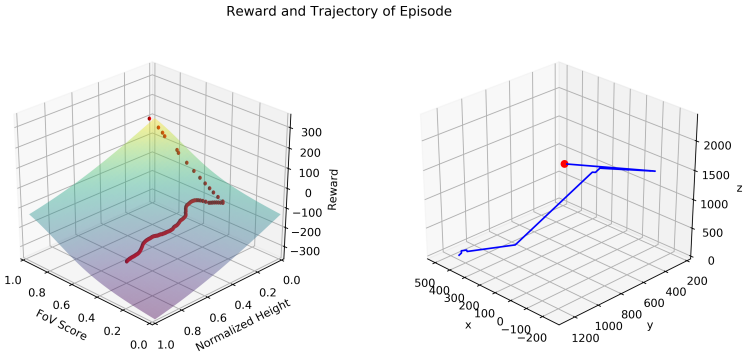


Figure 5.7: Left: The reward surface spanned by the FOV and height with the collected rewards shown as red spheres on the surface. Right: the trajectory taken by the the agent to land successfully. These two plots show a successful episode for a policy trained with early termination. The agent produces similar behaviour to fig. 5.3 but more reliably.

The reward signals of ten episodes performed by this policy show that the reward is generally increasing with each step as it progresses through the episode. A general upward slope is visible over all episodes, indicating that the policy is optimising for both FOV and Height.

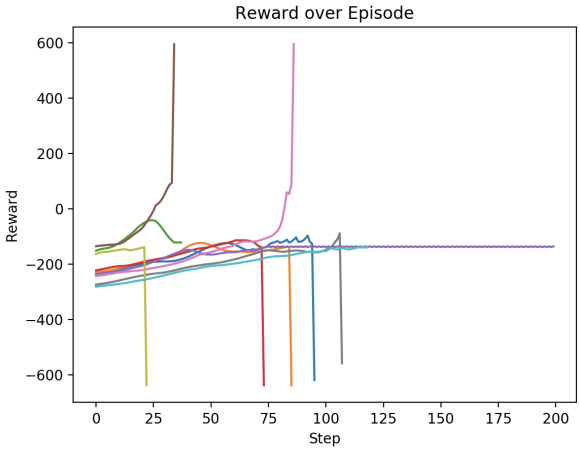


Figure 5.8: The reward signals of ten different episode for an agent using a policy trained in the above stated configuration. The reward signals have an upward slope throughout episode, indicating that both Height and FOV are being optimised for. Episodes end in either success or failure, with 2 exceptions.

5.4 Comparing to Random Agents

In order to benchmark the trained agents, a comparison was made to agents acting on a random subset of actions. Three random agents were used to benchmark the trained policies. The first purely random agent samples across all 27 actions uniformly. The second random agent, RandomDown agent, sampled across all actions that moved it downward or kept it in the same plane. The last random agent, JustLand agent, only moves straight down.

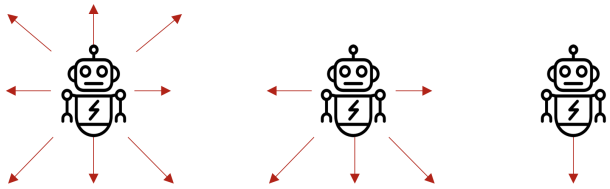


Figure 5.9: Illustration of movement directions of random agents.

fig. 5.10 shows the comparison between the random agents and the agents running the policies mentioned above. The policy trained with early termination produced the most successful landings out of the 50 attempts followed by the JustLand agent. The performance of JustLand agent is justified by the choice of scene used for the experiments and further experiments in other scenes would have to be conducted to evaluate the performance further. Nonetheless, the outperformance of the policy trained with early termination and no step penalty compared to all other agents proves that reinforcement learning can be used to produce intelligent policies that effectively land MAVs safely, albeit in simple settings.

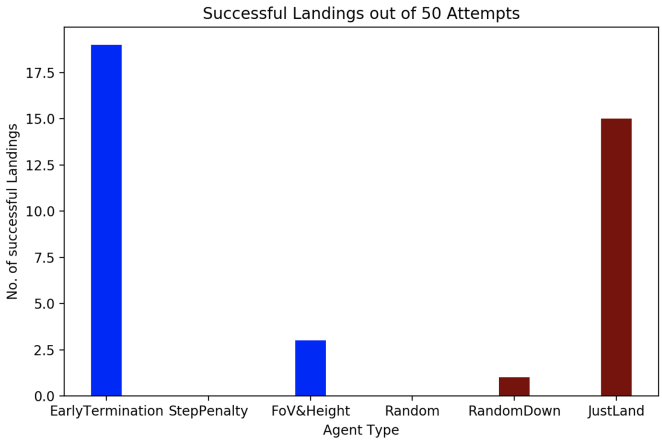


Figure 5.10: Comparison of performance of different agents vs. random agents. The bar plot indicates how many landings out of 50 attempts were successful.

Chapter 6

Conclusion

6.1 Discussion

This project aimed at contributing to the general framework used for developing reinforcement learning agents by extending and consolidating the existing ones. Establishing this framework allowed us to demonstrate the feasibility of using reinforcement learning for training an agent to land in safe landing sites and explore the effects of reward shaping on the overall outcome of the policy.

Regarding the general framework for reinforcement learning, it is important to note that this paradigm of machine learning is still in an early stage and there is no generally applicable framework that can be used to conduct research. As a result many of these tools must be implemented or adapted to suit the needs of the researcher, which are costly undertakings. There is a need for a common framework for interfacing with game engines and Open AI's Gym in order to streamline the research process. We believe that the framework proposed in this project may serve as a starting point for this framework that can be extended and improved over time.

A proof of concept (PoC) was demonstrated for the autonomous landing of MAVs using reinforcement learning. Although the demonstrated PoC is set in a simple scene with a simple model of a MAV, it illustrates many of the challenges one is faced with when training an agents policy. It may even be argued that the simpler settings provide better generalization of the policy. We believe that the findings presented do conclude the feasibility of the reinforcement learning approach. However, further research is necessary for finding an ideal reward shape and training setup for deriving a robust policy. The challenge remains in the near endless possibilities in how the reward can be formulated. The research conducted showed that introducing notions such as early termination in the training setup have notable effects on the policies performance. Furthermore, We attempted to illustrate some of the effects reward shaping can have on the end policy but also highlighted the shear number of degrees of freedom in training setup and reward shape.

6.2 Future Work

For future work, a parallelised version of the training setup needs to be implemented in order to reduce training time without sacrificing samples. Furthermore, the reward and training schemes presented need to be tested and trained in other environments to test their robustness. It is still unclear whether the best possible reward and training setup has been discovered, hence why more research needs to be

conducted on viable reward alternatives, since the performance achieved may serve as an early proof of concept, however it does not constitute a satisfactory performance standard. In future research, the feasibility of AirSim should be reevaluated in light of the new contributions made to the open-source environment. AirSim already includes many scene models and pawn models that can be mixed and matched to suit ones needs without having to get one's hands dirty with unreal editor. Policies could then be trained in more realistic scene settings and simulate more realistic agent dynamics. It is also important to note that the agent constructed in this work used a discrete action space and in future works the focus should lie on continuous action spaces to produce smoother agent behaviour.

Bibliography

[1] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 111–118.

[2] H. W. Ho, C. D. Wagter, and B. D. W. Remes, "Optical-flow based self-supervised learning of obstacle appearance applied to MAV landing," p. 17.

[3] T. Hinzmann, T. Stastny, C. Cadena, R. Siegwart, and I. Gilitschenski, "Free LSD: Prior-free visual landing site detection for autonomous planes."

[4] I. Epic Games. (2021) Unreal engine.

[5] Y. Z. S. Q. Z. X. T. S. K. Y. W. A. Y. Weichao Qiu, Fangwei Zhong, "Unrealcv: Virtual worlds for computer vision," *ACM Multimedia Open Source Software Competition*, 2017.

[6] (2021) Open ai gym.

[7] T. Y. A. Y. Y. W. Fangwei Zhong, Weichao Qiu, "Gym-unrealcv: Realistic virtual worlds for visual reinforcement learning," Web Page, 2017.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning series. The MIT Press.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms."

[10] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization."

[11] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation."

[12] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," 2020.

[13] (2021) Unity game engine.

[14] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles."

[15] M. J. Mataric, "Reward functions for accelerated learning," in *Machine Learning Proceedings 1994*. Elsevier, pp. 181–189.

Appendix A

Hyperparameters

Hyperparameters training

Parameter	Value
Hidden layer size	256
learning rate	3e-4
# of steps per iteration	20
minibatch size	5
epochs	4
max # of steps	15000

Hyperparameters GAE

Parameter	Value
λ	0.95
γ	0.99