# LEARNING FROM DEMONSTRATIONS: APPLICATIONS TO AUTONOMOUS UAV LANDING AND MINECRAFT

A Thesis

by

PRABHASA KALKUR

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

| | |
|---|---|
| Chair of Committee, | Dileep Manisseri Kalathil |
| Committee Members, | Srinivas Shakkottai |
| | Jean-Francois Chamberland |
| | Moble Benedict |
| Head of Department, | Miroslav Begovic |

December 2020

Major Subject: Electrical and Computer Engineering

# ABSTRACT

As robots and other intelligent agents move from simple environments to more complex, unstructured settings, modern deep reinforcement learning (RL) systems require an exponentially increasing number of samples for learning. This sample-inefficiency is not practical, especially for many real-world tasks when environment samples and compute are expensive. One way to address these limitations is to use sample-efficient methods like imitation learning (IL), that leverage human demonstrations to generate the desired behavior. IL has proven to be effective in problems with singular tasks, especially on tasks related to robotic manipulation and autonomous vehicles.

This Thesis applies existing state-of-the-art IL techniques to learn desired behaviors in two complex, sparsely-rewarded, simulated real-world systems: Unmanned Aerial Vehicles (UAVs) and video games. Specifically, we study the sample-efficiency and imitation accuracy of Generative Adversarial Imitation Learning and Deep Q-learning from Demonstrations on the two tasks respectively, as opposed to learning behavior purely from RL.

Current control systems do not capture the intuition and decision-making skills behind a pilot's maneuver, which can be crucial for landing under uncertainties. We design a novel method of sample-efficient autonomous UAV maneuver & landing using Microsoft AirSim as an environment simulator. The second application, motivated by the MineRL competition, involves learning to solve tasks like chopping trees and obtaining a diamond on Minecraft, using Microsoft Malmo as the environment simulator. For the problem of autonomous UAV maneuver & landing, we demonstrate sample-efficient imitation and comment on the need for 'smooth' experts for learning optimal landings. We show improved performance on pairing RL methods with demonstrations, over RL methods alone, for the problem of learning to chop trees on Minecraft. Further, we discuss design choices and potential bottlenecks from using the simulators and attempt to address them.

# DEDICATION

To my mother and father, who have always stood by me and have been my source of motivation and support.

# ACKNOWLEDGMENTS

This project would not have happened without the regular guidance and support from Prof. Dileep. Our collaboration with Prof. Benedict and his students got my Thesis work rolling. A big thank you for Prof. Dileep's great course on Reinforcement Learning (RL), which motivated me to pursue research in the area of imitation learning. Shout-out to Bochan Lee for helping set up experiments on Microsoft AirSim, and Vinicius Guimares Goecks for bearing with the meeting sessions and the barrage of emails. Kudos to all members of LeNS lab for creating a conducive environment for interesting discussions, both academic and otherwise. I would also like to thank Prof. Dileep and Prof. Shakkottai for organizing regular group meetings, and for their insightful comments on the talks. I thoroughly enjoyed and learned a great deal from the students' presentations. **Special thanks** to Kishan Panaganti Badrinath for helping with the MineRL competition and Sapana Chaudhary for being patient with me and providing valuable insights throughout the research (especially in a year like 2020!). Lastly, I would like to thank the Texas A&M University Office of Graduate and Professional Studies for allowing me to construct this LaTeX thesis template.

## CONTRIBUTORS AND FUNDING SOURCES

### Contributors

This work was supported by a thesis committee consisting of Professor Dileep Manisseri Kalathil [advisor] and Professors Srinivas Shakkottai and Jean-Francois Chamberland of the Department of Electrical and Computer Engineering and Professor Moble Benedict of the Department of Aerospace Engineering.

The human demonstrations for the AirSim environment was provided in part by Bochan Lee from the Department of Aerospace Engineering. All other work conducted for the thesis was completed by the student independently.

### Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

# NOMENCLATURE

| | |
|---|---|
| BC | Behavioral Cloning |
| DQfD | Deep Q-networks from Demonstrations |
| DQN | Deep Q-Networks |
| fD | from Demonstrations |
| GAIL | Generative Adversarial Imitation Learning |
| GAN | Generative Adversarial Networks |
| HPs | Hyperparameters |
| IL | Imitation Learning |
| IRL | Inverse Reinforcement Learning |
| ML | Machine Learning |
| MuJoCo | Multi-Joint dynamics with Contact |
| RL | Reinforcement Learning |
| SAC | Soft Actor-Critic |
| TRPO | Trust Region Policy Optimization |
| UAV | Unmanned Aerial Vehicles |

# TABLE OF CONTENTS

# LIST OF FIGURES

FIGURE Page

# LIST OF TABLES

TABLE Page

# 1. INTRODUCTION

## 1.1 Background

### 1.1.1 Machine Learning

Machine Learning (ML) deals with the study of algorithms used to find patterns in data. Over the last few decades, ML has seen a tremendous growth particularly due to the success of deep learning algorithms. Deep learning is a branch of ML that uses neural networks to represent and identify complex patterns in data.

The field of ML can be broadly classified into three parts: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning uses labeled data to learn patterns in data, e.g. algorithms that classify images as either a cat or a dog. Unsupervised learning attempts to discover patterns in unlabelled data, e.g. algorithms that cluster data based on their similarities. Reinforcement learning involves algorithms learning to take actions that maximize the notion of a cumulative reward of an environment, e.g. playing video games to maximize score.

Recent advances in hardware and hence compute have led to a massive growth in deep learning applications to academia and industry. This includes domains such as robotics, autonomous systems, biotechnology, natural language processing, computer vision, and lately, reinforcement learning.

### 1.1.2 Reinforcement Learning

Consider an agent continually interacting with an environment as shown in Figure 1.1, say a self-driving car passing through an intersection. The agent takes an **action** in the environment, say the car accelerates. It observes the environment at the current point in time, which is the **state** of the system and receives a positive or negative reinforcement signal from the environment, which we call the **reward**. In the case of the car, the system may penalize the car for getting too close to a pedestrian, or give a positive reward for speeding up, so the pedestrian can cross quickly on the car's passing. The car responds to the environment's feedback by taking appropriate action to transition to a new state, like braking if it received a negative reward or continuing to accelerate if it received a positive reward. This repeated process of the agent taking an action and observing a new state and reward can be considered performing a **task** in the system, with reaching a certain goal in mind. Intuitively, the goal is associated with the maximum reward, and thus our objective is to maximize the reward obtained over the duration of the task.

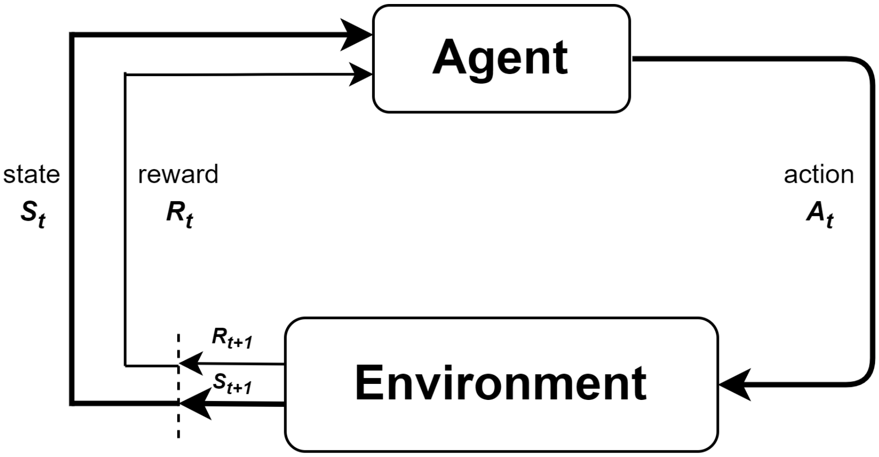


Figure 1.1: Reinforcement Learning: A primer.

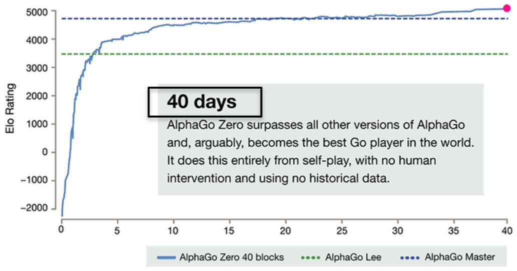
More formally, let $S, A$ be the collection of all possible states and actions for the task and consider $s, s' \in S,\ a \in A$. The transition of the system from the state $s$ to a new state $s'$ may be explained by a system model, represented as a transition probability $P(s'|s, a)$. Let $H$ be the horizon of the task. The rewards $R(s, a)$ received over time can be scaled by a discount factor $\gamma \in [0, 1]$, signifying the importance of future rewards over immediate rewards. Define a mapping $\pi : S \to A$, which we will call the policy of the task.

The tuple $[S, A, P(s'|s, a), R(s, a), \gamma, H]$, called a Markov Decision Process (MDP), represents the framework of learning from interaction to achieve a goal. The objective is to obtain an optimal policy $\pi^*$ that maximizes the quantity $\sum_{t=0}^{\infty} \gamma_t R_t$, where $R_t = R(s_t, \pi(s_t))$. In the case of unknown dynamics $P$, this problem is known as **Reinforcement Learning (RL)**. The class of algorithms that use a neural network to represent the policy fall under deep-RL.

## 1.2 Problem

### 1.2.1 Need for Sample-Efficiency

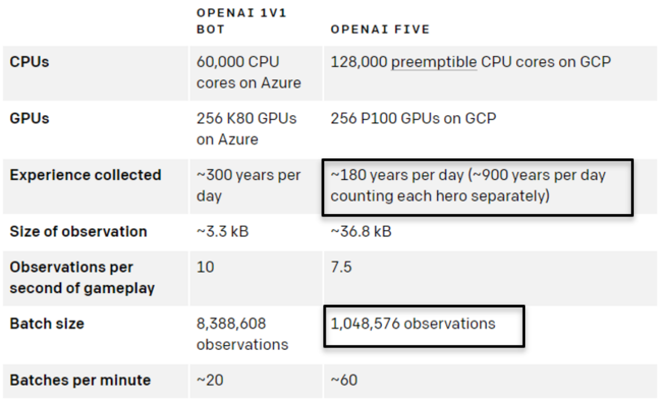
With an increase in problem complexity, deep-RL methods require a larger number of samples for training. Recent successes in Artificial Intelligence (AI) such as AlphaGoZero (4.9 million games of self-play) [1], OpenAI Five (11,000+ years of Dota 2 gameplay) [2], and AlphaStar (200 years of Starcraft II gameplay) [5] use deep-RL to achieve human or super-human level performance in sequential decision-making tasks. As shown in Figures 1.2 and 1.3, these tasks take massive amounts of training samples and compute. Several real-world applications such as self-driving vehicles and video games are affected by the sample requirements of such RL algorithms. This is not practical, especially when environment samples are expensive and compute is limited. It is also particularly challenging for the AI community to reproduce state-of-the-art results.



40 days

AlphaGo Zero surpasses all other versions of AlphaGo and, arguably, becomes the best Go player in the world. It does this entirely from self-play, with no human intervention and using no historical data.

Figure 1.2: AlphaGo Zero: Learning to play Go. Reprinted from [1].



| | OPENAI 1V1 BOT | OPENAI FIVE |
|---|---|---|
| CPUs | 60,000 CPU cores on Azure | 128,000 preemptible CPU cores on GCP |
| GPUs | 256 K80 GPUs on Azure | 256 P100 GPUs on GCP |
| Experience collected | ~300 years per day | ~180 years per day (~900 years per day counting each hero separately) |
| Size of observation | ~3.3 kB | ~36.8 kB |
| Observations per second of gameplay | 10 | 7.5 |
| Batch size | 8,388,608 observations | 1,048,576 observations |
| Batches per minute | ~20 | ~60 |

Figure 1.3: OpenAI Five: Learning to play Dota. Reprinted from [2].

### 1.2.2 Reward Design

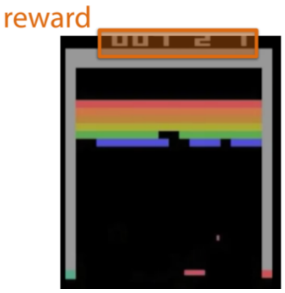


Figure 1.4: Rewards in computer games.

Figure 1.5: A self-driving car at an intersection.

Reinforcement Learning (RL) has been very successful in domains such as computer games. However, a key limitation of RL algorithms is that it involves the optimization of a predefined reward function or reinforcement signal [6–12], which, in the case of computer games, is to maximize the score as shown in Figure 1.4. However, there is no explicit reward function associated with many real-world tasks such as driving and robotic manipulation. Consider the example from earlier of a self-driving car passing through an intersection, as shown in Figure 1.5. Depending on the objective, there can be many reward functions for the task:

- Positive reward for slowing down at intersections to let the pedestrian cross first
- Positive/negative reward for speeding up, so the pedestrian can cross later
- Penalty for getting too close to the pedestrian

It is not obvious what the 'right' reward function is, and how we should weigh between the different components. Typically, we use a proxy reward function that does exactly that. However, manually designing such a reward function may lead to counter-intuitive behaviors [13].

It is often easier for humans to demonstrate a desired behavior for the task than try to manually engineer it, say, by careful design of reward functions. This need for sample-efficiency and for avoiding counter-intuitive behaviors resulting from manual reward function design motivates the problem of learning from readily available example expert behavior.

### 1.2.3 Imitation Learning

One approach to address the two issues is to use sample-efficient methods like imitation learning, that learn purely from demonstrations. Imitation learning (IL) is the study of techniques that learn from expert demonstrations, which are typically a collection of (state, action) pairs. IL methods have succeeded in addressing these issues in a wide range of problems [14–17], and have shown to be effective on tasks related to robotic manipulation and autonomous vehicles. For this thesis, we will apply (i) the state-of-the-art IL algorithm: Generative Adversarial Imitation Learning (GAIL) [3], and (ii) an algorithm that pairs RL with expert demonstrations: Deep Q-learning from Demonstrations (DQfD) [18] to two real-world tasks respectively.

For better understanding, figure 1.6 and Figure 1.7 roughly illustrate the difference between Reinforcement Learning (RL) and Imitation Learning (IL) techniques for learning.

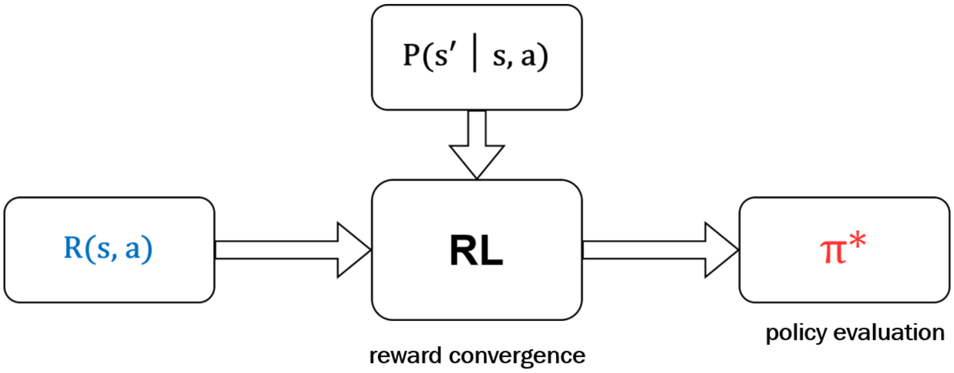


Figure 1.6: Reinforcement Learning.

In RL (the case where system dynamics $P$ is unknown), we obtain a policy that maximizes a given reward function for the task. To verify learning, we look at two performance metrics:

1. Evaluate the resulting policy and check if the policy can obtain the desired reward
2. The convergence of the policy reward to the desired reward, over the training phase. This is more of a sanity check but acts as a first pass to verify learning

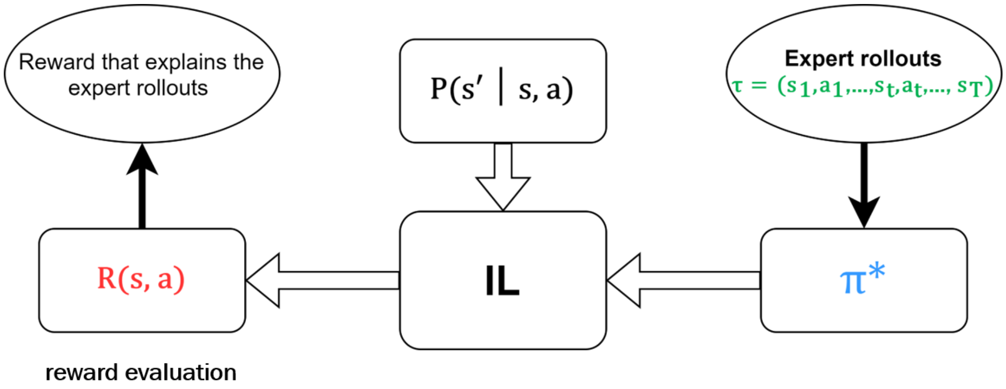

Reward that explains the expert rollouts
P(s′ | s, a)
Expert rollouts
τ = (s1,a1,...,st,at,..., sT)
R(s, a)
IL
π*
reward evaluation

Figure 1.7: Imitation Learning.

In IL, we typically obtain a reward that explains the given expert demonstrations. To evaluate the reward, we typically obtain a policy that maximizes this predicted reward. To verify imitation, we look at the same performance metrics as RL. However, this requires the construction of a proxy reward function if it is explicitly not available, as is the case for many real-world tasks.

The remainder of this thesis is structured as follows - we motivate our choice of GAIL as the IL algorithm in section 2. In section 3, we will study the imitation accuracy and sample-efficiency of GAIL with the help of a popular control task. We then apply GAIL to the task of autonomous Unmanned Aerial Vehicle (UAV) maneuver and landing in section 4. For performing tasks in Minecraft in section 5, we demonstrate the benefits of using RL with demonstration data, as opposed to just RL. Finally, in section 6, we summarize the conclusions drawn from our experiments and discuss possible improvements and extensions to the applications.

Our goal is to learn expert behavior using as few demonstrations as possible, which is the goal of sample-efficient learning. In this section, we discuss some popular imitation learning (IL) methods and motivate our choice of Generative Adversarial Imitation Learning (GAIL).

## 2.1 Early Approaches to Imitation Learning

One of the earliest attempts at learning behavior aimed at purely ‘mimicking’ expert action, without worry about the expert dynamics. This approach later termed Behavioural Cloning (BC), focuses on learning the expert’s behavior as a supervised learning problem. However, BC does not generalize well and can run into failures due to compounding errors. Direct Policy Learning improves on BC by assuming access to an expert and knowledge of system dynamics during training. However, these assumptions are not practical for many real-world applications such as self-driving vehicles and playing video games. Abbeel and Ng [14] later introduced apprenticeship learning, also known as Inverse reinforcement learning (IRL), which recovers a reward function for the task to learn behavior. However, this is an under-defined problem, as there may be many reward functions that can explain expert behavior. We would thus like to recover a unique reward function.

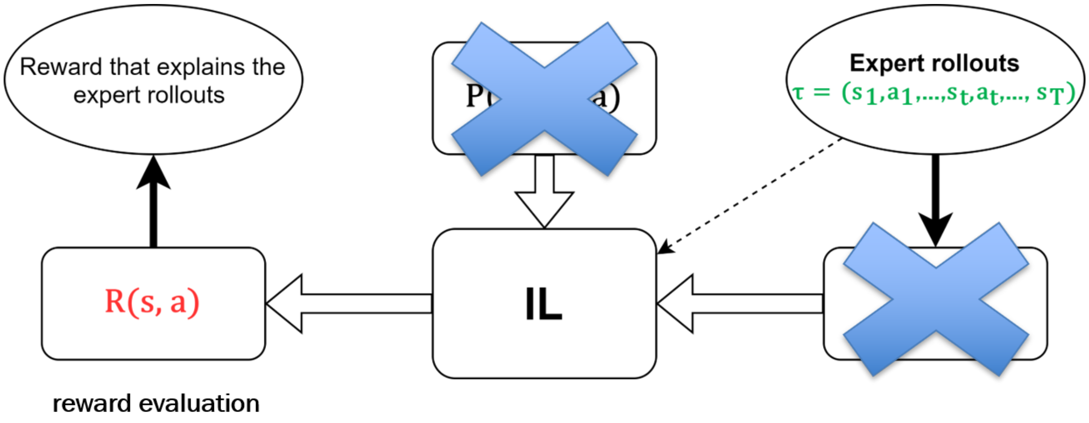

Reward that explains the expert rollouts
Expert rollouts
$\tau = (s_1, a_1, ..., s_t, a_t, ..., s_T)$
R(s, a)
IL
reward evaluation

Figure 2.1: Inverse Reinforcement Learning.

The modified block diagram of IL from Section 1, shown above in Figure 2.1, is IRL. The dotted arrow describes the problem of learning purely from expert demonstrations, without access to the expert (crossed). We will also assume that we do not have access to the dynamics of the system, which is the case in most real-world tasks.

Ziebart et al. [15] proposed a maximum entropy principle for IRL, which assigns a probability distribution for sampling the expert demonstrations. That is, demonstrations with higher rewards are more likely to be sampled, and this prior assumption helps recover a unique reward function. However, evaluating the recovered reward in IRL still requires running RL after each update, which is a large overhead. Rewriting the flowchart from earlier as a complete imitation learning framework of IRL, we have Figure 2.2.

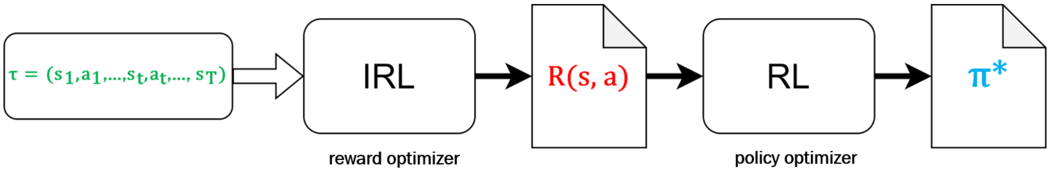


Figure 2.2: Imitation Learning via IRL.

## 2.2 Generative Adversarial Imitation Learning

Our objective is to extract expert behavior from demonstrations. That is, we would like to recover the expert policy. Recovering the reward as an intermediate step is somewhat of an overhead. Ho and Ermon [3] proposed Generative Adversarial Imitation Learning (GAIL) that directly recovers a policy from expert demonstrations, "as if it were obtained by RL following IRL."

GAIL is a model-free IL algorithm that directly learns a policy from expert behavior, without any access to a reinforcement signal or expert feedback. The training process of GAIL can be thought of as building a generative model, whose goal is to generate behaviors similar to the expert rollouts, which are a collection of (state, action) pairs. It does this by evaluating its stochastic policy on a fixed simulation environment.

Imitation is achieved by jointly training a discriminator to distinguish expert trajectories from ones produced by the learned policy, just as in GANs [19]. GAIL can be seen as optimizing max-entropy IRL with a special form of discriminator network, which is the same as optimizing the discriminator in GANs. If the generator network fools the discriminator into thinking the rollouts came from the expert, then we say GAIL has learned to imitate the expert.

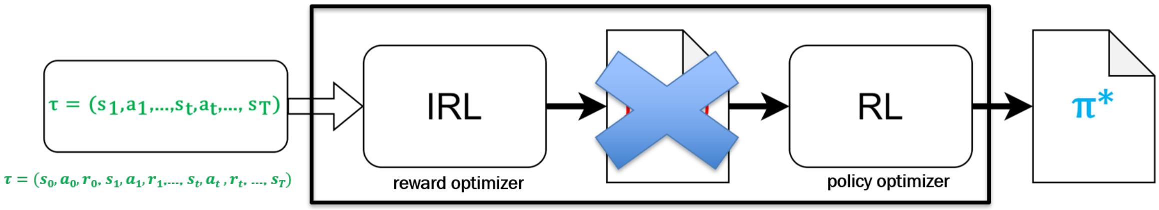

$\tau = (s_1, a_1, ..., s_t, a_t, ..., s_T)$
$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_t, a_t, r_t, ..., s_T)$
IRL
reward optimizer
RL
policy optimizer
$\pi^*$

Figure 2.3: Generative Adversarial Imitation Learning.

The authors show that GAIL can imitate complex behaviors in large, high-dimensional environments, which is our motivation behind applying it to real-world tasks like autonomous UAV landing. Using the IRL framework from earlier, GAIL can be roughly seen as the black box shown in Figure 2.3. The algorithm is as shown in Figure 2.4 below.

Some additional pointers on GAIL:

- Trust Region Policy Optimizer [8] (TRPO) is used as the policy optimizer
- If there is an available/proxy reward function, this can be included as part of the demonstrations in order to use reward convergence as a performance metric

```
Algorithm 1 Generative adversarial imitation learning
Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters $\theta_0, w_0$
for $i = 0, 1, 2, \ldots$ do
    Sample trajectories $\tau_i \sim \pi_{\theta_i}$
    Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient
        $\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))]$    (17)
    Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$.
    Specifically, take a KL-constrained natural gradient step with
        $\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$
        where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$    (18)
end for
```

Figure 2.4: Generative Adversarial Imitation Learning. Reprinted from [3].

To summarize the imitation learning methods as shown in Figure 2.5, GAIL is the state-of-the-art IL algorithm that addresses several challenges faced by earlier methods. We will use GAIL to demonstrate the sample-efficiency of imitation learning, by learning behaviors purely from demonstration data of a real-world task.

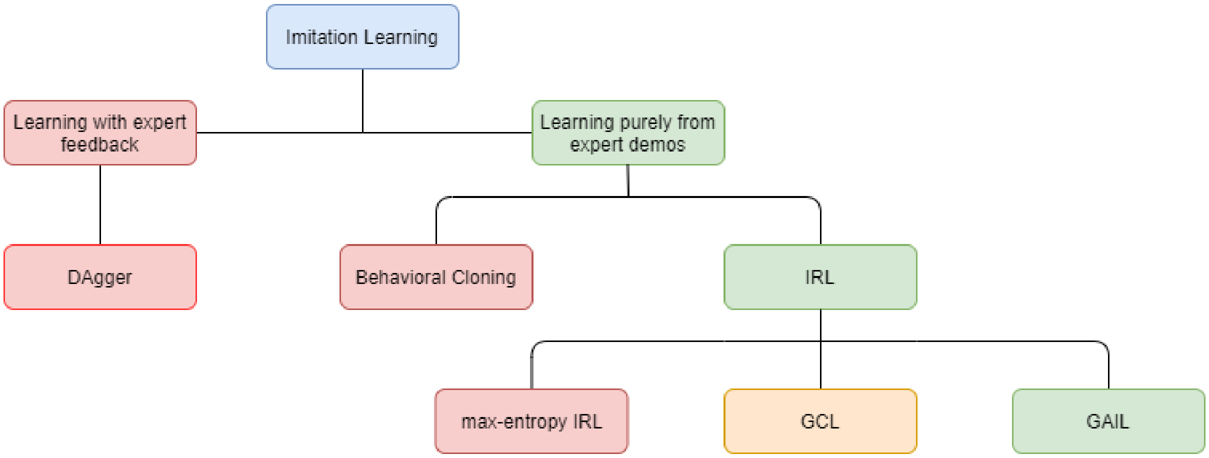


Figure 2.5: Imitation Learning methods: a summary.

# 3. PROBLEM SETUP

Our goal is to extract a policy (learn behavior) purely from demonstration data of a task. Depending on the availability of expert demonstrations, we train IL algorithms (GAIL) on demonstrations of the task as follows:

1. Expert data available: Sample expert -> Train GAIL -> Evaluate policy
2. Expert data unavailable: Train RL on available / proxy reward function -> Rollout experts -> follow step 1

## 3.1 Performance Metrics

**Expected outcome:** GAIL should be able to learn the expert policy, for both optimal and suboptimal experts. We call the learned policy a "successful imitation" of the expert if the statistics for the two are similar, The similarity is quantified using the two performance metrics mentioned in Section 1. Listed below are some keywords related to learning the expert policy:

1. Expert rollouts: Generate 100 trajectories of performing the task and sample [5, 10, 20] trajectories to train GAIL on. The expert trajectories may be optimal or suboptimal.
2. Policy evaluation: Generate rollouts from the learned policy for 100 episodes, and check if it solves the task
3. Task solved: Learned policy achieves true reward for 100 consecutive episodes of training
4. Successful imitation: If the reward statistics from policy evaluation match that of expert rollouts

We will mostly use two **performance metrics to check for imitation:**

1. Reward convergence: mean cumulative episode reward vs episodes / env interactions
2. Policy evaluation: demonstrations used vs (mean, std) episode reward

### 3.1.1 Tools Used

- **RL Library:** Stable Baselines 2.10 [20]
- **Framework:** TensorFlow 1.14
- **Hyperparameters (HPs):** RL Baselines Zoo [21]
- **Performance metrics:** Tensorboard 1.14, W&B 0.10 [22]

For our experiments, we will borrow the implementation of RL algorithms, BC, and GAIL from Stable Baselines 2.10. We will make use of tuned hyperparameters (HPs) available on RL Baselines Zoo [21] for experiments on the case studies and the first application. Note that this thesis will only list the HPs borrowed from Zoo, and all other HP values are the default values found on Stable Baselines 2.10.

We also use some features and techniques such as multiprocessing for speed-up, data normalization for high-dimensional tasks, and periodic policy evaluation to monitor performance during training. We will not divulge into the details, but using these features resulted in a significant boost in time and performance in many experiments.

### 3.1.2 Interesting Questions

To understand the policy (behavior) learned by GAIL, we will use popular control tasks from OpenAI Gym. These simulated tasks do not have readily available expert data, and we will use Soft Actor-Critic (SAC) [12], an RL algorithm, to learn an optimal policy, from which we rollout experts. GAIL can then be trained on the expert rollouts to learn a policy (behavior), from which we will try to answer the following questions:

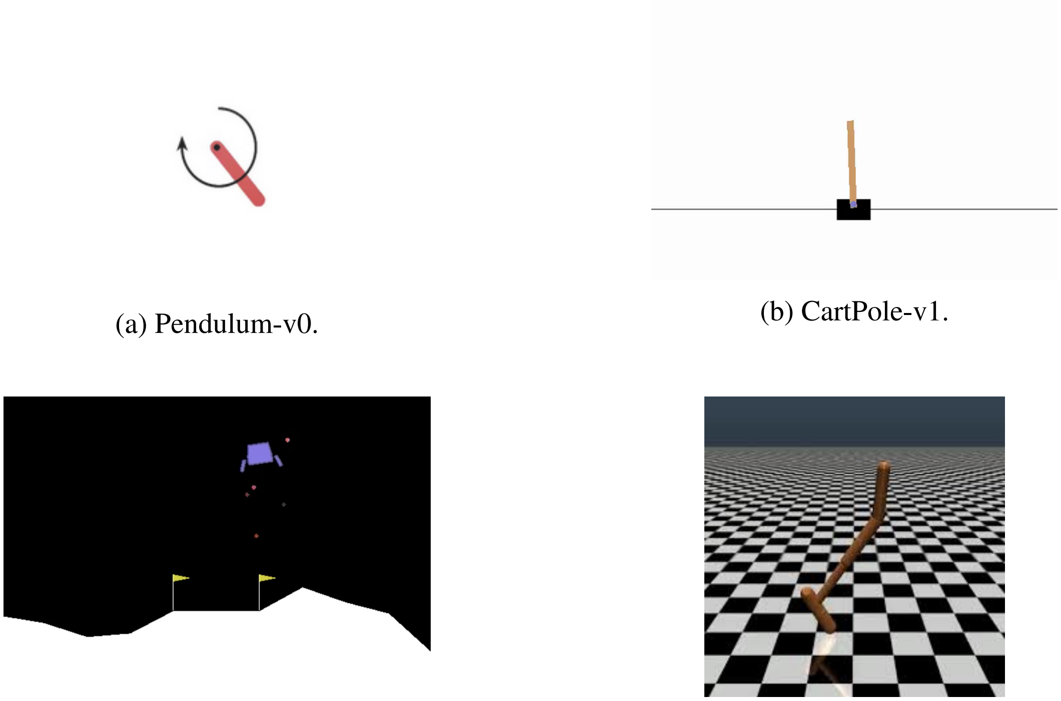
1. How does imitation accuracy scale with problem dimensionality (complexity) and demonstration data (sample-efficiency)?
2. How 'smooth' are the learned policies compared to the expert policy?
3. Can sparse rewards be learned? At what cost?
4. Can GAIL imitate suboptimal experts?

### 3.2 OpenAI Gym and MuJoCo

OpenAI Gym [23] is a “Toolkit for developing and comparing reinforcement learning algorithms.” It is a platform for teaching agents to perform simulated tasks under a true reward. This means that expert demonstrations can be generated for these tasks by training SAC can on the true reward. Some classes of tasks on Gym are Atari games, robotic manipulation, and classic control tasks such as balancing a pendulum and a cart pole upright.

MuJoCo [24] is “a physics engine that does very detailed, efficient simulations with contacts.” This includes several 2D and 3D continuous control tasks such as hopping, walking, and running. These platforms are used as standard benchmark tasks for testing RL, IL algorithms in the literature.

We will use control tasks from these platforms to draw conclusions about GAIL’s imitation accuracy and sample-efficiency. We will later notice similarities between our applications and a control task, which can be used to explain behavior learned for one of our applications. Table 3.1 lists some common hyperparameters used for GAIL throughout the thesis.



(a) Pendulum-v0.

(b) CartPole-v1.

(c) LunarLander-v2.

(d) Hopper-v2.

Figure 3.1: Control tasks from OpenAI Gym and MuJoCo.

| **GAIL Hyperparameters** | **Value** |
|---|---|
| | |
| Policy (both Disc & Gen) | 'MlpPolicy' |
| Discriminator NN architecture | $[100, 100]$ |
| Generator NN architecture | $[100, 100]$ |
| Activation, initial weights | Tanh, 1 |
| Optimizer | Adam, learning rate $3 \times 10^{-4}$ |
| Loss function: Discriminator | Binary Cross-Entropy (BCE) |
| Batch size | $64$ |

Table 3.1: GAIL hyperparameters common throughout the thesis.

### 3.3 Case Study: GAIL on Pendulum-v0

To analyze the imitation accuracy and sample-efficiency of GAIL, we consider the OpenAI Gym control task of keeping an inverted pendulum upright (Pendulum-v0). The details of the environment are listed below in Table 3.2:

| **Properties** | **Description** |
|---|---|
| | |
| State space (cts, dim=3) | Cosine, sine of angle $\theta$ $[-1, 1], \theta_0$ $[-8, 8]$ |
| Action space (cts, dim=1) | Joint effort [-2, 2] |
| Reward | - ($\theta^2 + 0.1 \times \theta_0^2 + 0.001 \times$ action$^2$) |
| Termination | 200 steps |
| Solved criteria | No solved criteria. Defined as -200 |
| Expert trajectories used | [5, 10, 20] randomly sampled from 100 demonstrations |

Table 3.2: The Pendulum-v0 environment.

We run SAC on the true reward for the task and rollout experts from the resulting policy. We sample [5, 10, 20] expert trajectories from the rollouts and run GAIL. The hyperparameters (from RL Zoo) used for both SAC and GAIL are listed in Table 3.3. We use two performance metrics to check for imitation:

1. Reward convergence: if the expert is optimal, cumulative reward converges
2. Policy evaluation: episode statistics must match that of the expert. Statistics include reward (mean, variance) and episode length (mean, variance)

Fig. 3.2 shows GAIL's reward over training, on different sample sizes of the expert. We observe that the mean reward has converged to the true reward (-200), and conclude that GAIL learned the Pendulum-v0 task. However, this does not guarantee imitation of the expert.

| **Algorithm** | **Hyperparameters** | **Value** |
|---|---|---|
| | | |
| SAC | env interactions | $6 \times 10^4$ |
| | policy | 'MlpPolicy' |
| | learning starts | 1000 |
| | | |
| GAIL | env interactions | $3 \times 10^5$ |
| | policy | 'MlpPolicy' |
| | max KL | 0.000193 |
| | timesteps per batch | 1024 |
| | gamma | 0.99 |
| | lambda (GAE) | 0.98 |
| | entropy coefficient | 0.01118 |
| | cg damping | $2.35 \times 10^{-5}$ |
| | value function iterations | 10 |
| | value function stepsize | 0.00428 |

Table 3.3: Hyperparameters for SAC and GAIL on Pendulum-v0.

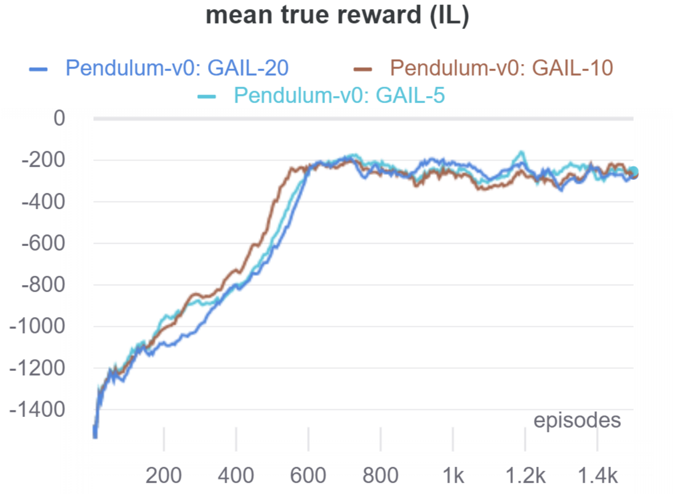
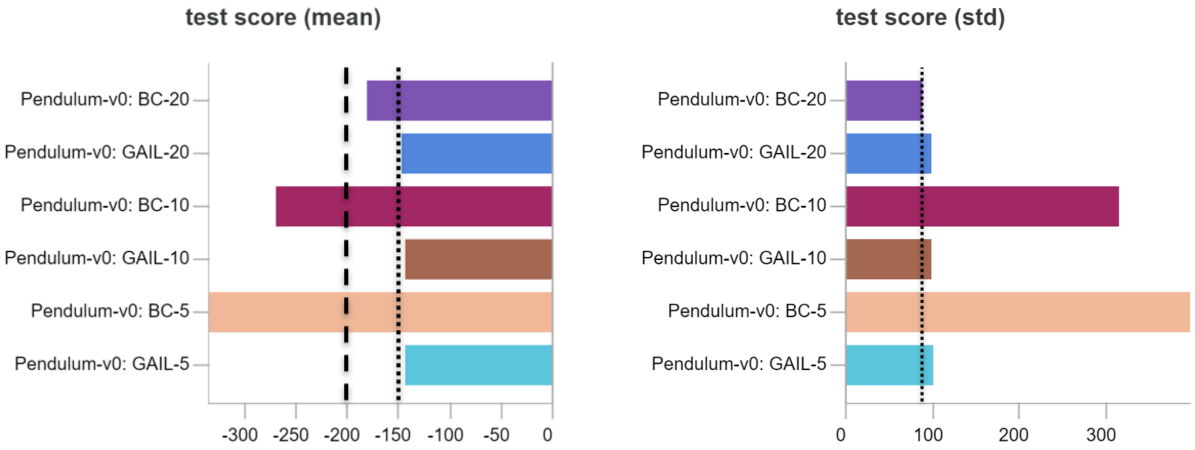


Figure 3.2: GAIL on Pendulum-v0: Reward convergence over training

Fig. 3.3 shows the policy evaluation of the trained GAIL, BC models for different numbers of expert demos. The dashed lines represent the true reward of the task, and GAIL's policy evaluation corroborates the reward convergence. The dotted line represents expert statistics. It is interesting to note that the learned policy's performance is not only consistent over the number of demonstrations but also matches that of the expert. That is, GAIL needs only 5 demonstrations to learn to perform the task just like the expert and performs similar to learning from 10 or 20 demonstrations. GAIL can imitate the Pendulum-v0 expert behavior while being sample-efficient.



(a) Episode reward (mean).

(b) Episode reward (std).

Figure 3.3: GAIL on Pendulum-v0: policy evaluation, with episode reward (mean, std). Dashed line shows the true reward, dotted lines show the expert rollout statistics

Note that the performance of Behavioral Cloning improves with demo data, but takes more than 20 demonstrations to imitate expert performance. More importantly, this improvement is only observed on using optimal experts and fails when suboptimal experts are used. Meanwhile, GAIL is also able to successfully imitate suboptimal experts, as shown on the CartPole-v1 task below in Table 3.4. This gap in performance between BC and GAIL only widens with task complexity, hence we will not use BC as a comparison algorithm for our applications.

| **Data** | **Optimal** | **Reward** | **Length** | **Success rate** |
|---|---|---|---|---|
| | | | | |
| Expert rollouts | No | **(402, 134)** | 402 | 63/100 |
| Learned policy | No | **(410, 178)** | 411 | 59/100 |

Table 3.4: GAIL on 10 suboptimal CartPole-v1 rollouts: policy evaluation.

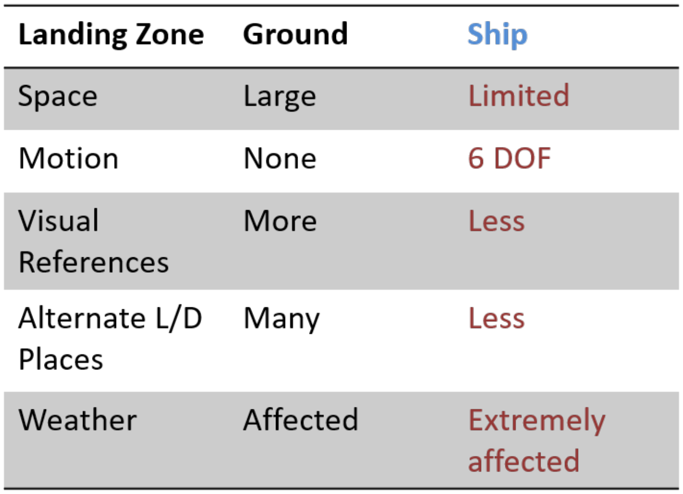
The reward convergence and policy evaluation of the learned policies showed “successful imitation” of the expert. Let us now answer the questions from before:

1. How does imitation accuracy scale with problem dimensionality and demo data? **GAIL is sample-efficient on low-dimensional control tasks, BC is not**
2. How ‘smooth’ are the learned policies compared to expert policy? **smooth if the episode reward (std) on policy evaluation matches the episode reward (std) of expert rollouts**
3. Can sparse rewards be learned? At what cost? (we only looked at tasks with dense rewards)
4. Can GAIL imitate suboptimal experts? **For low-dimensional control tasks, yes**

To summarize, GAIL successfully imitates two low-dimensional control tasks, in the case of both optimal and suboptimal experts. With this background, we look at the problem of autonomous UAV maneuver and landing, a more complex, sparsely-rewarded task. We will try to answer the same set of questions, and analyze GAIL’s imitation accuracy and sample-efficiency on the task.

# 4. APPLICATION: AUTONOMOUS UAV MANEUVER & LANDING

This application was inspired by the procedure that Navy pilots typically follow for landing on space-constrained ship decks. As shown in Figure 4.1, landing a helicopter on a moving ship is a hard problem and requires more skill than landing on solid ground. It is hard for the pilot to physically see the landing pad, and pilots typically refer to a gyro-stabilized horizon reference bar (a visual cue) for approaching the ship deck [25, 26], as shown in Figure 4.2.



| Landing Zone | Ground | Ship |
|---|---|---|
| Space | Large | Limited |
| Motion | None | 6 DOF |
| Visual References | More | Less |
| Alternate L/D Places | Many | Less |
| Weather | Affected | Extremely affected |

Figure 4.1: Comparing helicopter landing on the ground and a ship.

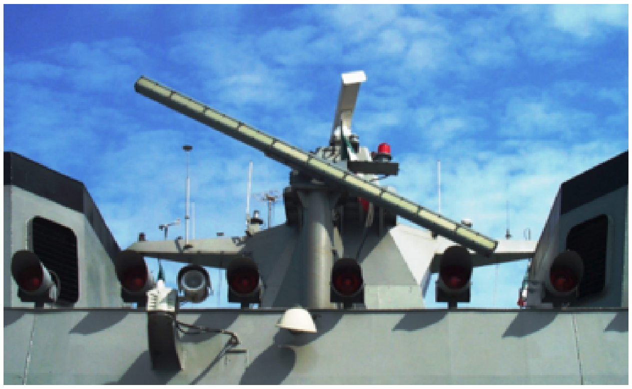
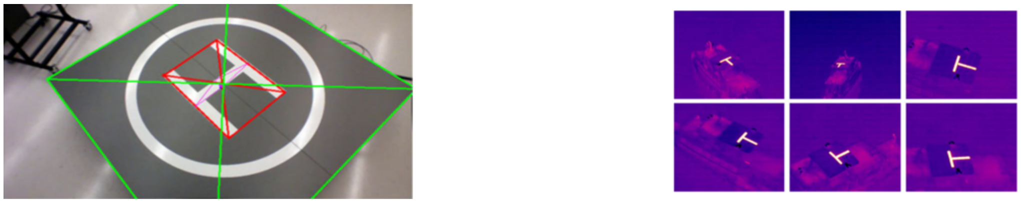


Figure 4.2: Gyro-stabilized reference bar used for approach and landing.



(a) Landing pad from [27]

(b) Capturing images from [28]

Figure 4.3: Vision-based autonomous landing techniques

Current autonomous systems employ vision-based methods for landing [27, 28], where the landing pad position and heading are captured using an on-board camera from the helicopter, as shown in Figures 4.3a and 4.3b. However, these control systems do not capture the pilot's intuition and decision-making skills, which is critical for handling uncertain scenarios. Human demonstrations are readily available for approach and landing based on the visual cue. This work proposes an imitation learning-based approach for autonomous Vertical Take-Off and Landing (VTOL) of Unmanned Aerial Vehicles (UAVs). Our contributions are threefold:

- Landing a UAV on a ship without looking at landing spot (in simulation)
- Referring to a visual cue for maneuver and positioning
- Bring pilot's intuition and flying skills with imitation learning

## 4.1 Environment Simulator

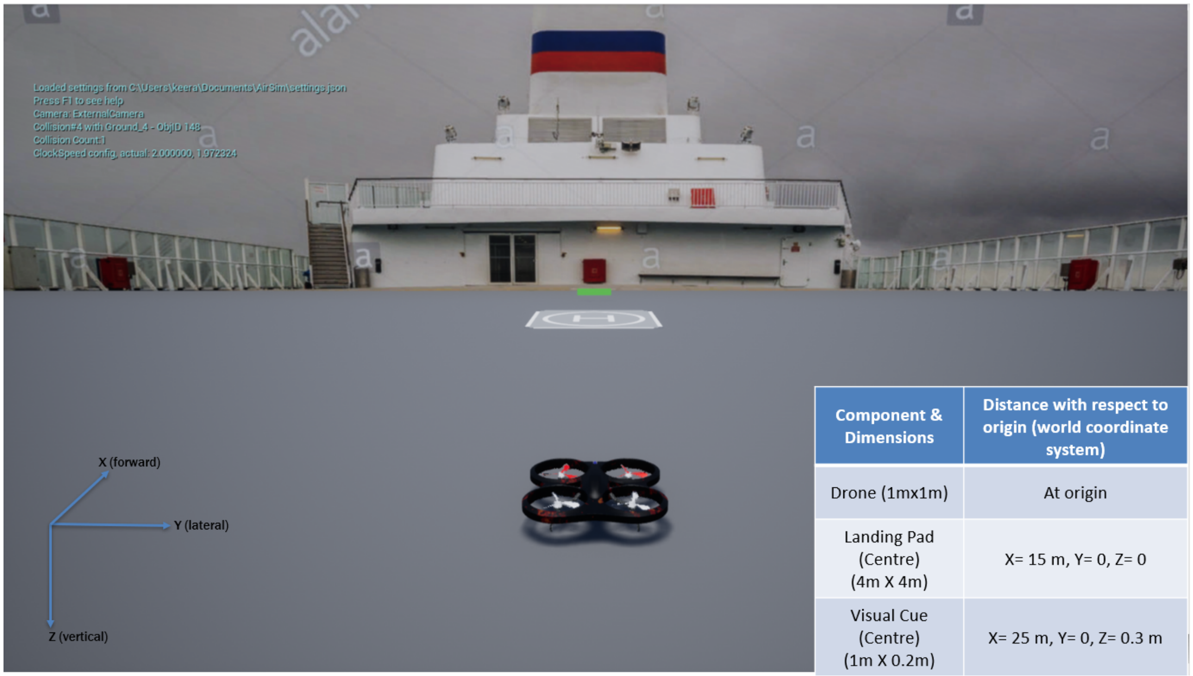
To simulate the problem of autonomous VOTL UAV landing, we will need to fly a UAV in a high-fidelity simulator environment. For this work, we choose Microsoft AirSim 2.0 [29]. It is an open-source, cross-platform simulator built on Unreal Engine [30].

### 4.1.1 Setting up The Simulator

AirSim has a multirotor vehicle that can be used as a UAV. For our experiments, we design a custom ship environment that reflects a standard ship deck, as seen in Figure 4.4 and Figure 4.5:

1. Drone: spawns at a random position within a bounding box, around the drone's origin
2. Landing pad (4m x 4m): 10 meters away from the drone in the forward direction

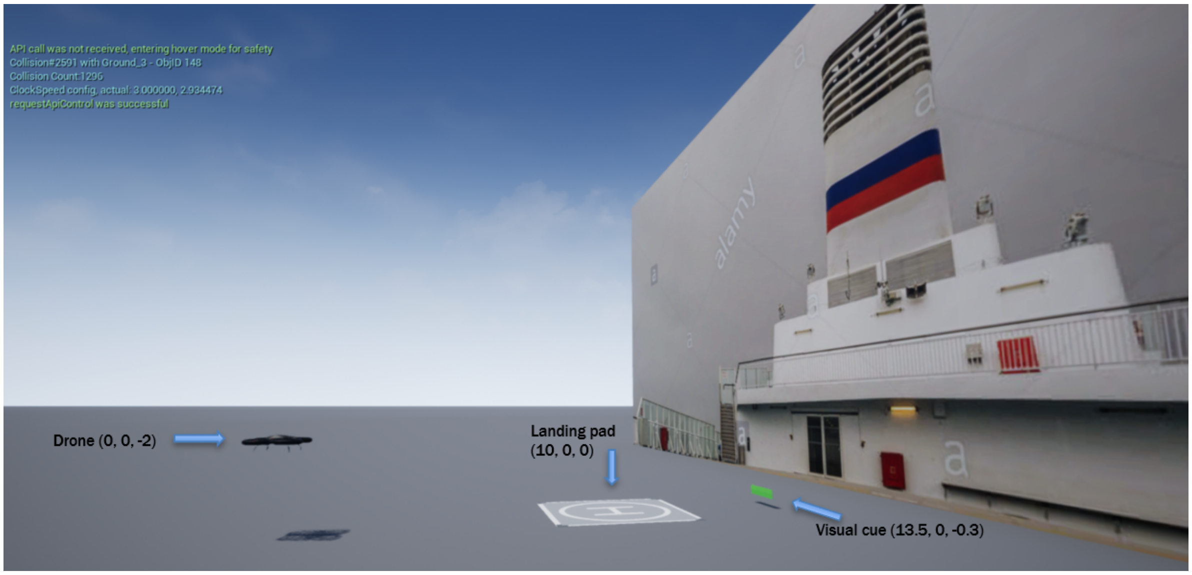
3. Visual cue: A green bar 13.5m away and 1.3m high from the drone in the forward direction



```
Loaded settings from C:\Users\keera\Documents\AirSim\settings.json
Press F1 to see help
Camera: ExternalCamera
Collision#4 with Ground_4 - ObjID 148
Collision Count:1
ClockSpeed config, actual: 2.000000, 1.972324
```

| Component & Dimensions | Distance with respect to origin (world coordinate system) |
|---|---|
| Drone (1mx1m) | At origin |
| Landing Pad (Centre) (4m X 4m) | X= 15 m, Y= 0, Z= 0 |
| Visual Cue (Centre) (1m X 0.2m) | X= 25 m, Y= 0, Z= 0.3 m |

Figure 4.4: AirSim setup: front view.



```
API call was not received, entering hover mode for safety
Collision#2591 with Ground_3 - ObjID 148
Collision Count:1296
ClockSpeed config, actual: 3.000000, 2.934474
requestApiControl was successful
```

Figure 4.5: AirSim setup: side view.

| **Properties** | **Description** |
|---|---|
| | |
| State space (cts, dim=6) | Drone position, velocity (X, Y, Z). **Goal:** 4x4 square<br>Position: X [0, 17], Y [-2, 2], Z [-5, 0] – negative Z upwards<br>Velocity: X [-1, 3], Y [-1, 1], Z [-4, 4] |
| Action space (cts, dim=3) | [Pitch (rad), Roll (rad), Throttle (0, 1)]. Yaw zero |
| Termination | Timeout, out of bounds, below visual cue, crash, land |
| Solved criteria | A proxy reward of 1000 |
| Expert trajectories used | [5, 10, 20, 50, 100] randomly sampled from demonstrations |

Table 4.1: The AirSim-v0 environment.

Table 4.1 lists the properties of the custom AirSim-v0 Gym environment. The drone's position and velocity constitute the state of the environment, and the drone's pitch, roll, and throttle are the actions that can be taken by the drone to reach a new state in the environment. Our goal is to learn a policy that performs the same maneuvers as the expert.

There are several factors for terminating the episode, including a timeout criteria. This timeout depends on the length of the expert demonstrations, and for purposes of generalizing, **will not assume a timeout criteria** for the moment, resulting in a variable-length horizon task. We note that there exists a similar OpenAI Gym control task without a timeout criteria (LunarLanderContinuous-v2), whose objective is also to land a rover on the ground.

### 4.1.2 Collecting Expert Demonstrations

AirSim can also integrate a variety of flight controllers. This feature can be used for collecting expert demonstrations. We calibrate a Taranis x9d flight controller shown in Figure 4.7, which allows the expert to map the joystick and enable/disable specific controls. To avoid any increase in problem dimensionality, we disable the yaw and focus only on the pitch, roll, and throttle of the drone. The maneuver to be made is as shown in Figure 4.6, where the drone refers to the visual cue for approach and is then allowed to fall below the visual cue for landing.

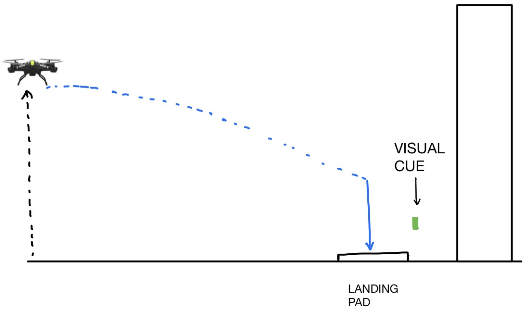


Figure 4.6: Intended maneuver by the expert.

Figure 4.7: Flight controller: collecting demos.

To quantify the expert data as optimal or suboptimal, and later quantify successful imitation, we design a proxy reward for the trajectories based on the following rule:

- Increase reward as it gets close to landing pad (1/x)
- A reward of +1000 on reaching the landing pad
- -10 for falling below visual cue before reaching pad, going out of bounds (termination)

The reward to be achieved for solving the task is 1000. Note that this is a **sparsely-rewarded** task, as there is a large positive reward for the goal state, which is roughly $10^5$ times larger than the small reward received at all other states in the environment.

We use the flight controller to generate sets of 120 optimal and 140 suboptimal expert demonstrations, shown in Figure 4.8 and Figure 4.9 respectively. The demonstrations follow the Stable Baselines requirements and constitute five components: (states, actions, rewards, returns, episode start). The state includes the drone's position and velocity, the actions include pitch, roll, and throttle. A video of the human expert recording the demonstrations can be seen here.

As mentioned in Section 2, rewards can be included in the expert demonstrations, in order to be able to use reward convergence as a performance metric. The reward also helps classify the expert as optimal or suboptimal, based on the reward needed to solve the task. Table 4.2 lists the statistics of the two experts. Based on the reward statistics, we observe that the second set of demonstrations is suboptimal due to its large variance in reward, which is visible on rendering the demonstrations. Note that the landing pad for the two sets of experts is centered at (10, 0, 0) and (15, 0, 0) respectively.

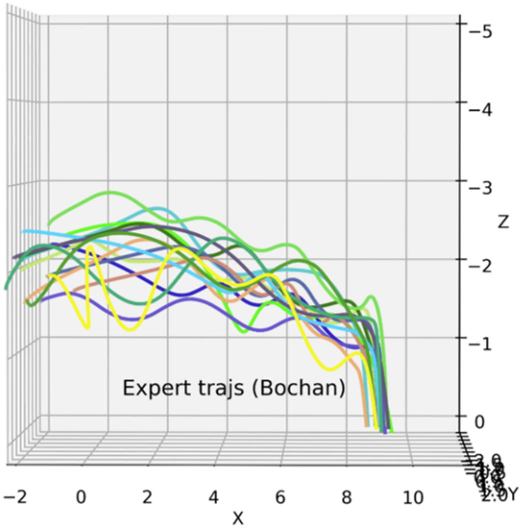


Figure 4.8: Optimal expert demos.

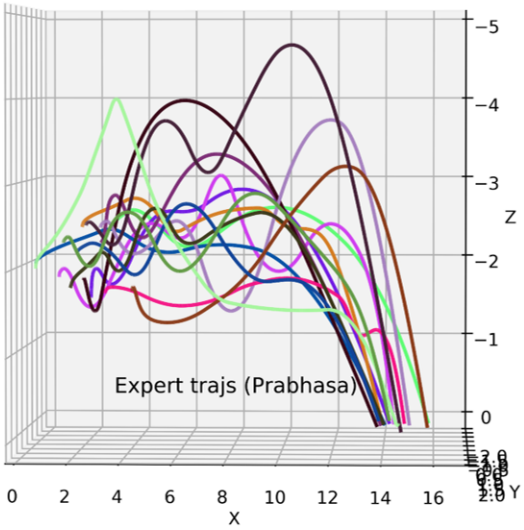


Figure 4.9: Suboptimal expert demos.

| **Quantity** | **Optimal** | **Suboptimal** |
|---|---|---|
| | | |
| Optimal | Yes | No |
| Solved | 120/120 | 132/140 |
| Expert score (mean) | 1141 | 1116 |
| Expert score (std) | 27 | **284** |
| Expert length | 362 | 307 |

Table 4.2: Human expert demonstrations - statistics.

To summarize, we train GAIL on the collected expert demonstrations and interact with the AirSim-v0 environment to learn the task of approach and landing. The reward needed to solve the task is 1000. For the experiments, we use the set of HPs available for the LunarLanderContinuous-v2 environment on RL Zoo, which we will not explicitly mention here. We observe that on using 5, 10 demonstrations GAIL is unable to solve the task, likely due to lack of data. We will thus use 20, 50, and 100 demonstrations for learning the maneuvers. A rendering of the expert demonstrations and training process can be seen here (**links:** expert demos, training phase - front view).

### 4.2 Results

The goal is to extract the expert's policy (learn maneuvers) by training GAIL on expert samples of size [20, 50, 100] from both optimal and suboptimal demonstrations. On using 20 or more optimal demonstrations, GAIL learned to approach the landing pad. However, it learned to hover over the landing pad until termination instead of landing, as shown in the video here. This could be due to the cumulative non-zero rewards received above the landing pad, which may be comparable to the goal state reward. To justify this behavior, we look at the Lunar Lander control task from Gym.

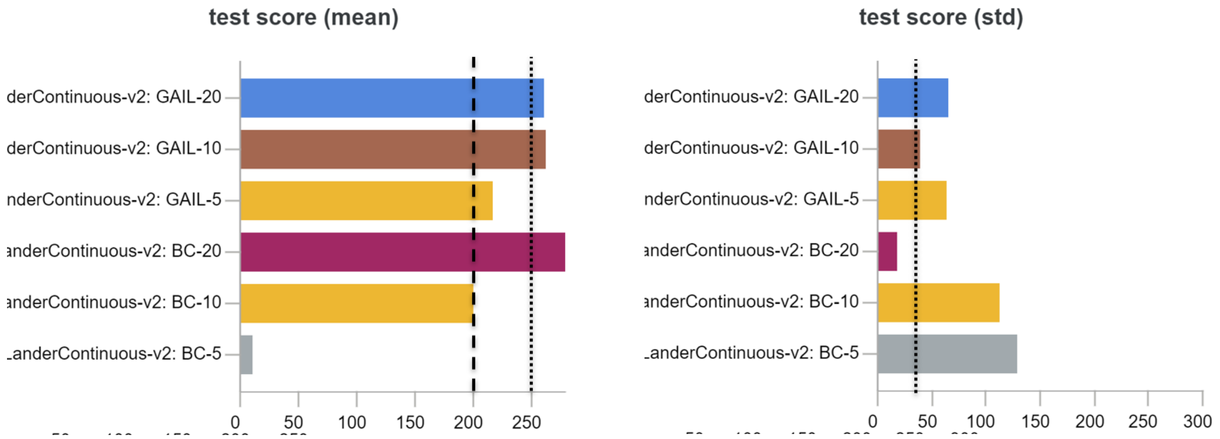
#### 4.2.1 Case Study: GAIL on LunarLanderContinuous-v2

The LunarLanderContinuous-v2 environment is as described in Table 4.3, which is quite similar to AirSim-v0. There is also no explicit termination criteria other than the goal state, which is our motivation for the case study.

Figure 4.10 shows the policy evaluation of the trained GAIL, BC models for different numbers of optimal demonstrations. The dashed lines represent the true reward of the task, and the dotted line represents the expert statistics. GAIL learns to solve the task for all pairs of demonstration data and also learns to imitate the LunarLanderContinuous-v2 expert behavior while being sample-efficient (more than 10 demonstrations).

| **Properties** | **Description** |
|---|---|
| | |
| State space (cts, dim=6)<br>and (disc, dim=2) | Lander position, velocity, angle & angular speed, leg contact<br>Position: X [-1, 1], Y [0, 1.4]. **Goal**: b/w flags at [-0.2, 0.2]<br>Velocity: typically [-1, 1]. Angle [-0.4, 0.4] in radians |
| Action space (cts, dim=3) | Engine throttle [main engine, left-right engines] |
| Reward | 1. Lander crashes or comes to rest: -100 or +100<br>2. Each leg ground contact is +10<br>3. Firing the main engine is -0.3 points each frame |
| Termination | Land (+100) or crash (-100), no termination |
| Solved criteria | Mean reward 200 over 100 consecutive episodes |
| Expert trajectories used | [5, 10, 20] randomly sampled from 100 demonstrations |

Table 4.3: The LunarLanderContinuous-v2 environment.

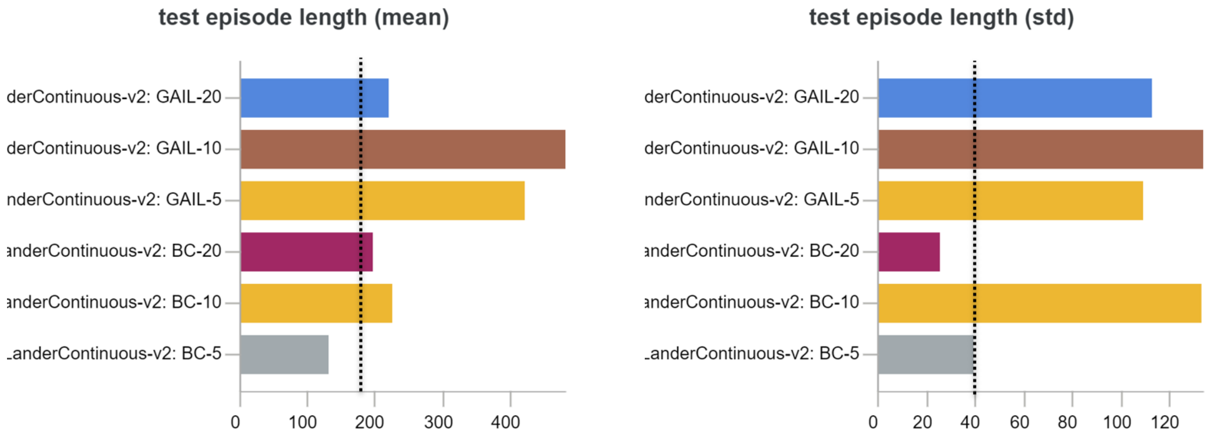


(a) Episode reward (mean).

(b) Episode reward (std).

Figure 4.10: GAIL on LunarLanderContinuous-v2: policy evaluation, with episode reward (mean, std). Dashed line shows true reward, dotted lines show expert rollout statistics

However, if we are to look at the episode length of the learned policy in comparison to the expert, there is a large difference between the two. Figure 4.11 shows the expert length statistics in dotted lines, clearly indicating that the learned policy learns longer episodes. On rendering, we learn that the learned policy lands the rover on the ground, but keeps one of its engines turned on to achieve a higher reward. That is, **GAIL learns to exploit the long horizon of the task to improve the score** to match the expert's score. This confirms our suspicion that the learned policy hovers above the landing pad to improve its score.



(a) Episode length (mean).

(b) Episode length (std).

Figure 4.11: GAIL on LunarLanderContinuous-v2: policy evaluation, with episode length (mean, std). Dotted lines show the expert rollout statistics

### 4.2.2 GAIL on Optimal Expert

For the remainder of the results, we choose to **fix the problem horizon** as the mean length of the expert demonstrations, which is 400. Figure 4.12 shows the average reward convergence of GAIL on using 20, 50, 100 optimal demonstrations. Although it falls short of the true reward represented by the dashed lines, it learns a policy similar to that of the optimal expert, as shown in Table 4.4. Figure 4.13b shows the maneuvers performed by the learned policy, as compared to the optimal expert shown in Figure 4.13a. GAIL learns an "average" behavior of the optimal expert.

| Data | Optimal | Reward | Length | Success rate |
|---|---|---|---|---|
| | | | | |
| Expert rollouts | Yes | **(1141. 27)** | 362 | 120 / 120 |
| Learned policy | Yes | **(1048, 292)** | 80 | 84 / 100 |

Table 4.4: GAIL on 20 optimal AirSim-v0 rollouts: policy evaluation.

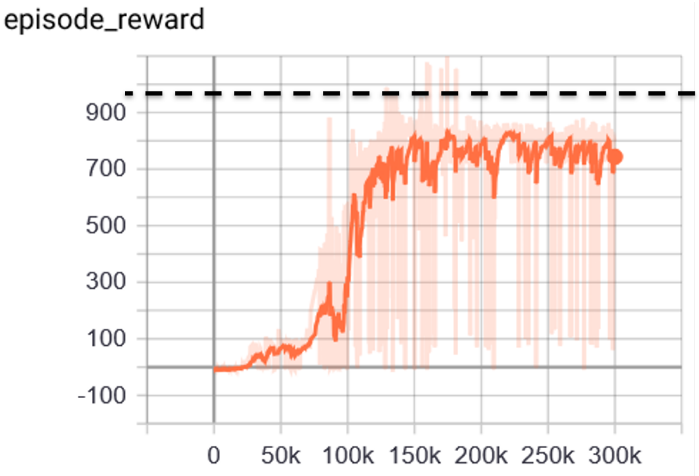
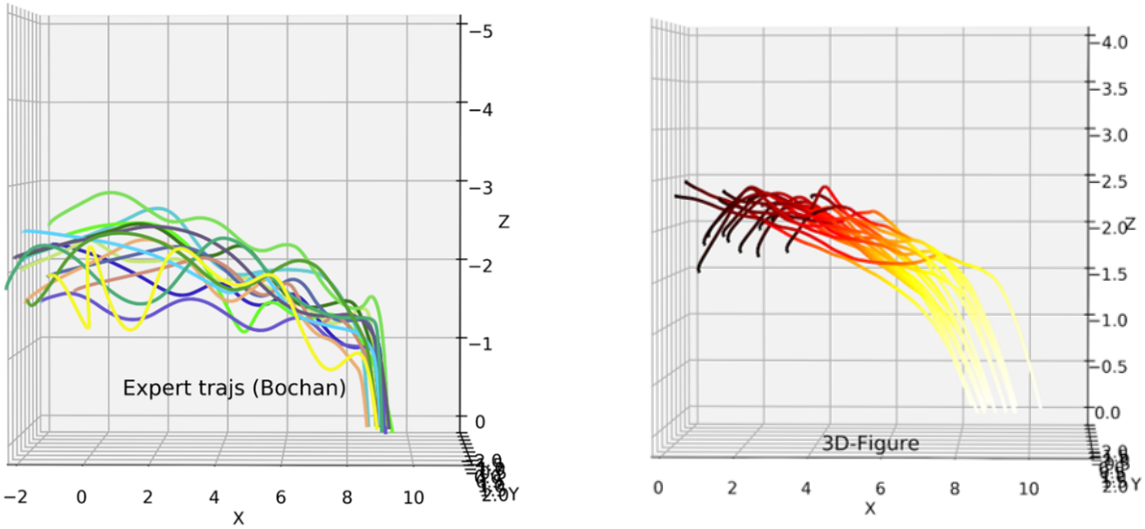


Figure 4.12: GAIL on optimal AirSim-v0 expert: reward convergence.

On careful look at rollouts from the learned policy as shown in the video here, we see that although the approach is similar to the expert, the landing is not perfect. The optimal expert performed hard landings, whereas the learned policy landings are smoother and sometimes land outside the landing pad. This results in reduced performance. We will address this issue in detail later.



(a) Optimal expert.

(b) Learned policy.

Figure 4.13: Optimal expert rollouts vs learned policy rollouts.

### 4.2.3 GAIL on Suboptimal Expert

Figure 4.14 shows the average reward convergence of GAIL on using 20, 50, 100 suboptimal demonstrations. Although it falls short of the true reward represented by the dashed lines, it learns to approach the landing pad as seen in the video here. Table 4.5 compares the performance of the learned policy with the suboptimal expert. Figure 4.15b shows the maneuvers performed by the learned policy, as compared to the optimal expert shown in Figure 4.15a. GAIL learns an "average" behavior of the approach, but the lack of learning to land is likely due to the large variance in the demonstration data.

| **Data** | **Optimal** | **Reward** | **Length** | **Success rate** |
|---|---|---|---|---|
| | | | | |
| Expert rollouts | No | **(1116. 284)** | 307 | 132 / 140 |
| Learned policy | No | **(684, 580)** | 80 | 12 / 100 |

Table 4.5: GAIL on 20 suboptimal AirSim-v0 rollouts: policy evaluation.

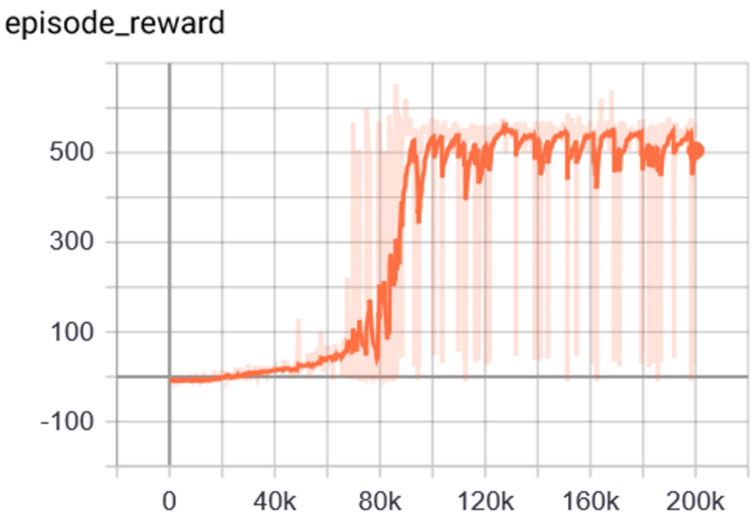
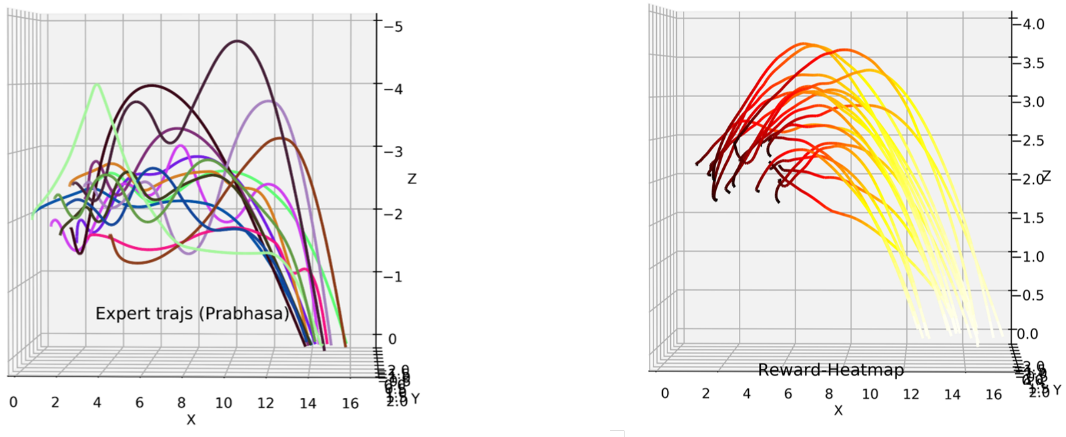


Figure 4.14: GAIL on suboptimal AirSim-v0 expert: reward convergence.



(a) Suboptimal expert.

(b) Learned policy.

Figure 4.15: Suboptimal expert rollouts vs learned policy rollouts.

## 4.3 Imitation Accuracy

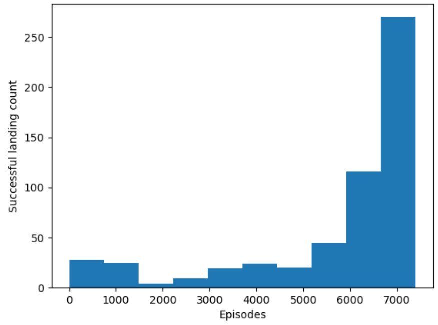


Figure 4.16: Histogram of successful landings over training.

Figure 4.16 shows increase in the frequency of landings over training. However, there is still a significant portion of unsuccessful landings. To further understand the gap between the problem of approach and the problem of landing, we will compare the different components of the state for rollouts of both the optimal expert and the learned policy. That is, we look at the **position and speed of the expert and learned policy over time**, for both approach and landing. Note the difference in timestep (x-axis) between the expert and learned policy are partly due to different sampling frequencies used for collecting expert data and training the GAIL policy.

### 4.3.1 Position Profiles

Shown below in Figure 4.17 and Figure 4.18 are the position profiles of the expert rollouts (left) and the learned policy (right), in the x-direction and the z-direction respectively. GAIL learns an "average" behavior of the optimal expert's approach.

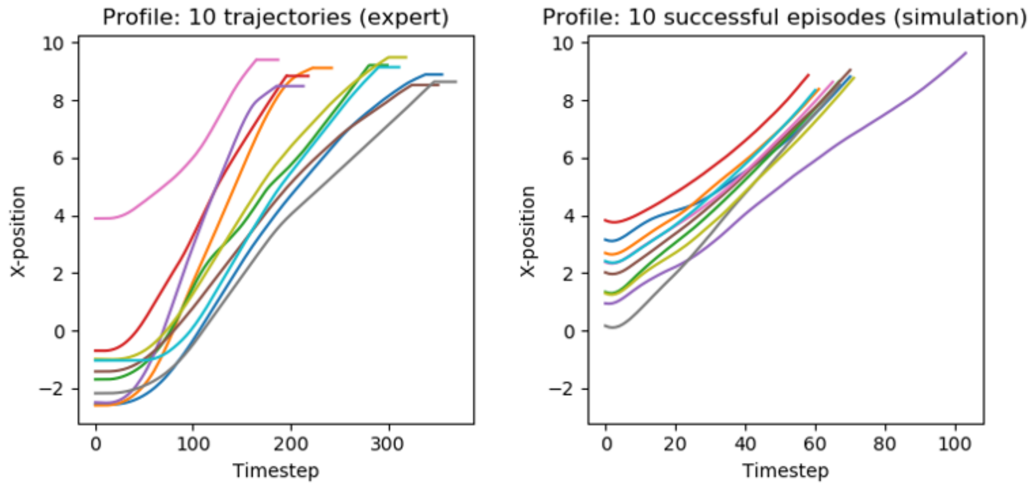


Figure 4.17: Position profile in the forward direction (X).

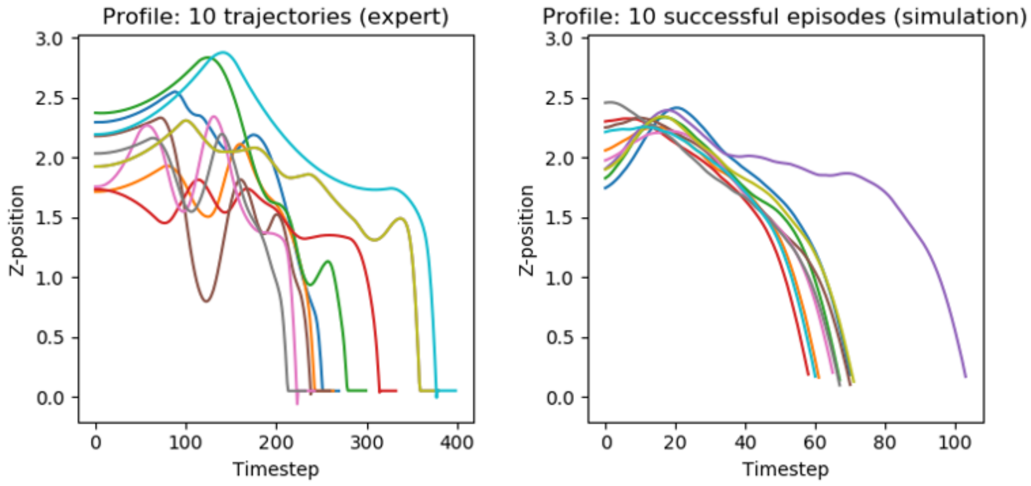


Figure 4.18: Position profile in the descent direction (Z).

### 4.3.2 Velocity Profiles

Shown below in Figure 4.19 and Figure 4.20 are the velocity profiles of the expert trajectories (left) and the learned trajectories (right), in the x-direction and the z-direction respectively. The optimal expert performs a "hard" landing, as seen by the sharp drop in velocity on reaching the goal state. This sharp, non-smooth transition in state is a difficult maneuver for GAIL to learn. Instead, it crashes on to the landing pad without slowing down.

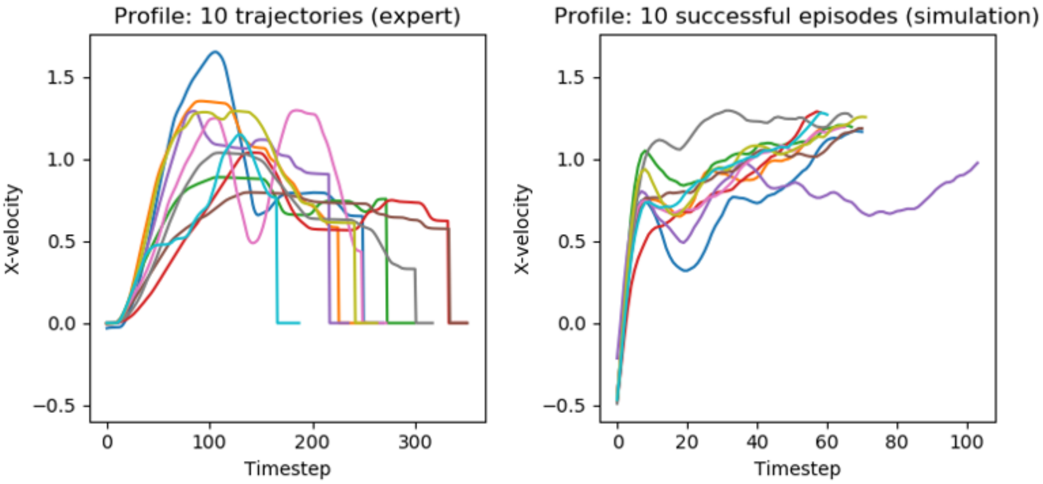


Figure 4.19: Velocity profile in the forward direction (X).

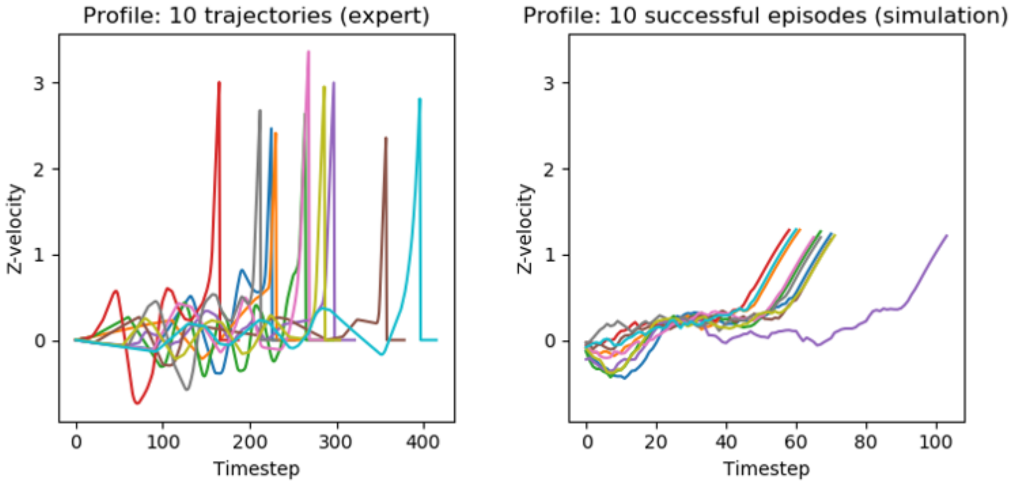


Figure 4.20: Velocity profile in the descent direction (Z).

The **non-smoothness of expert data** makes it hard for GAIL to learn to land. Although against the motivation for using GAIL, can we design a reward function that explains smoother landings than the human expert?

## 4.4 Proxy Reward Design

We look at the problem of designing two proxy reward functions that can generate smoother landings than the human expert, also resulting in improved performance. For this, we build on the proxy reward structure used earlier to classify the two experts as optimal and suboptimal. We will run Soft Actor-Critic (SAC), an RL algorithm, on these reward functions and observe their performance.

### 4.4.1 Proxy 1: Simple Reward Design

As our first proxy, we use the same reward function used to classify the expert data as optimal or suboptimal:

- Increase reward as it gets close to landing pad (1/x)
- A positive reward of +1000 on reaching the landing pad
- -10 for falling below visual cue, going out of bounds

On running SAC on this reward function, we can achieve the true reward, as shown by the performance metrics in Table 4.6 and Figure 4.21. As expected from the proxy, SAC learns to take the shortest path between the starting position and the landing pad, seen in Figure 4.22 and the video here.

| Data | Optimal | Reward | Length | Success rate |
|---|---|---|---|---|
| | | | | |
| Expert rollouts<br>(for comparison) | Yes | **(1141. 27)** | 362 | 120 / 120 |
| Learned policy | Yes | **(1106, 112)** | 99 | 99 / 100 |

Table 4.6: SAC on proxy reward 1: policy evaluation.

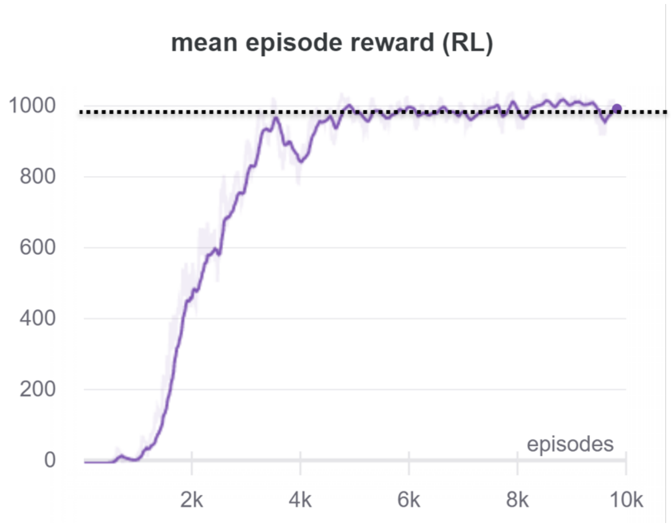


Figure 4.21: SAC on proxy reward 1: reward convergence.

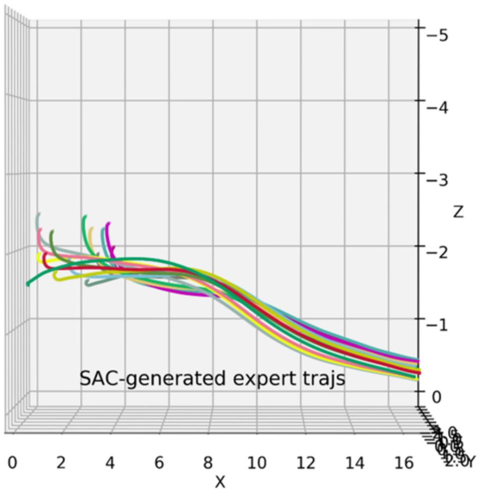


Figure 4.22: Learned policy rollouts.

### 4.4.2 Proxy 2: Complex Reward Design

For our second proxy, we scale the reward for landing with a factor based on the UAV's heading and speed. The idea is to bring in variation in speed so that the drone is rewarded more for slowing down, and not crashing into the landing pad:

- Increase reward as it gets close to landing pad (1/x)
- A scaled positive reward of +1000 on reaching the landing pad (1000*[1-scale, 1+scale])
- -10 for falling below visual cue, going out of bounds

On running SAC with this reward function, we can do better than the desired reward, as shown by the performance metrics in Table 4.7 and Figure 4.23. Note that the reward design was such that it can score more than the true reward only if it slowed down and adjusted its heading on landing. As expected from the proxy, SAC learns to make a more controlled maneuver, seen in Figure 4.24 and the video here. It is worth noting that the maneuver is visually similar to that of the optimal human expert.

| **Data** | **Optimal** | **Reward** | **Length** | **Success rate** |
|---|---|---|---|---|
| | | | | |
| Expert rollouts<br>(for comparison) | Yes | **(1141. 27)** | 362 | 120 / 120 |
| Learned policy | Yes | **(1265, 225)** | 94 | 97 / 100 |

Table 4.7: SAC on proxy reward 2: policy evaluation.

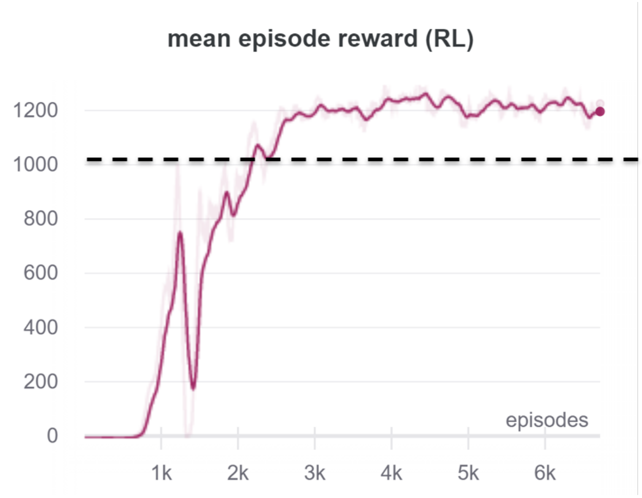


Figure 4.23: SAC on proxy reward 2: reward convergence.

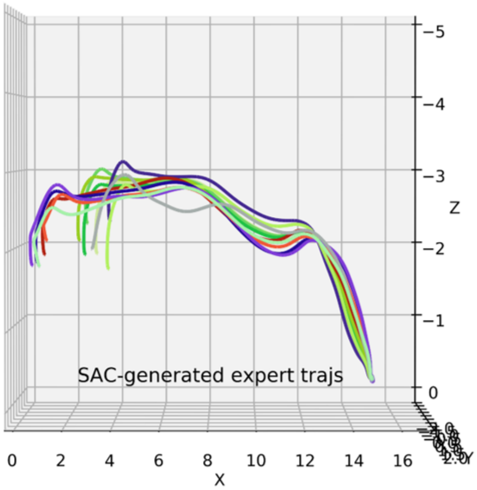


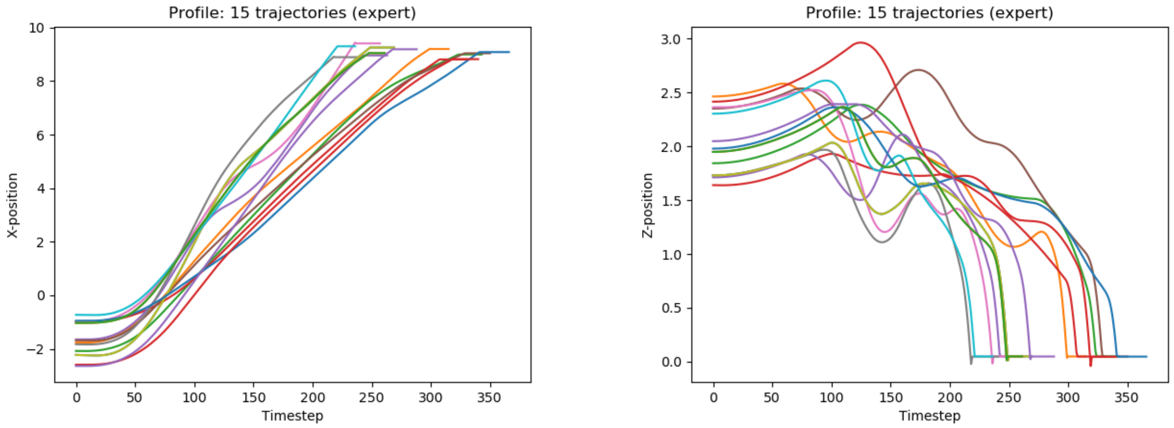
Figure 4.24: Learned policy rollouts.

## 4.5 The Gap Between Human Expert and RL

The complex reward proxy is able to learn maneuvers similar to the optimal expert. The more important distinction to be made is in the manner of landing. While the pilot typically makes hard landings on ships, its non-smoothness makes it a hard maneuver for GAIL to learn. However, the landings learned from the proxy reward are much smoother, and can be an easier task for GAIL to learn. This difference is also evident in the analysis of profiles, as seen earlier.

### 4.5.1 Position Profiles

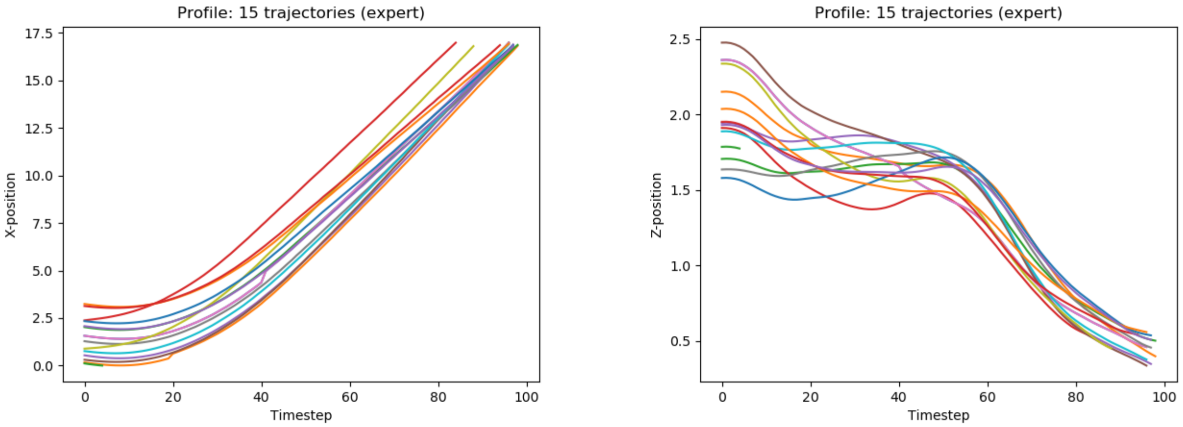
Shown below are the forward (left) and descent (right) profiles of the optimal human expert and the SAC-learned policy. The SAC-learned landings are much smoother, although less like what a pilot would do.



(a) X-direction: forward.

(b) Z-direction: descent.

Figure 4.25: Optimal human expert: position profiles.



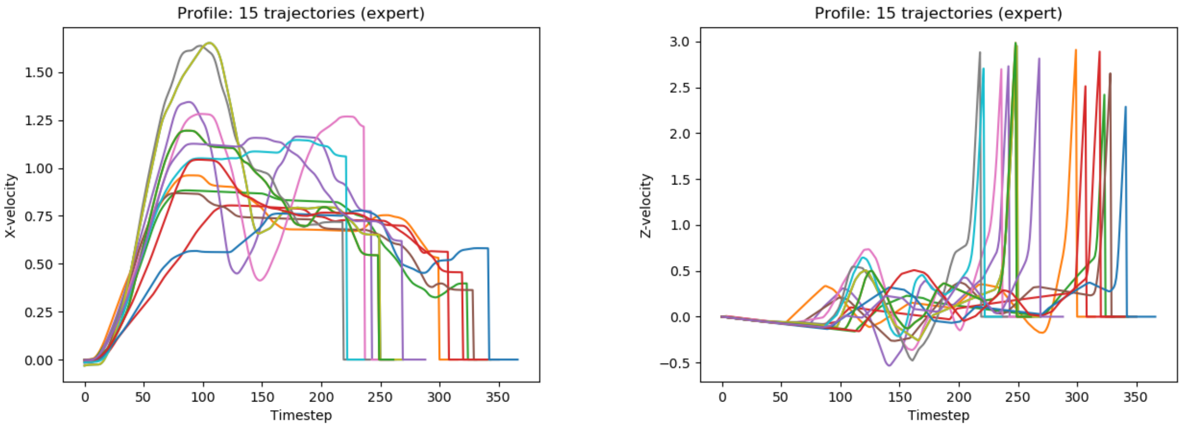
(a) X-direction: forward.

(b) Z-direction: descent.

Figure 4.26: SAC-learned policy: position profiles.

### 4.5.2 Velocity Profiles

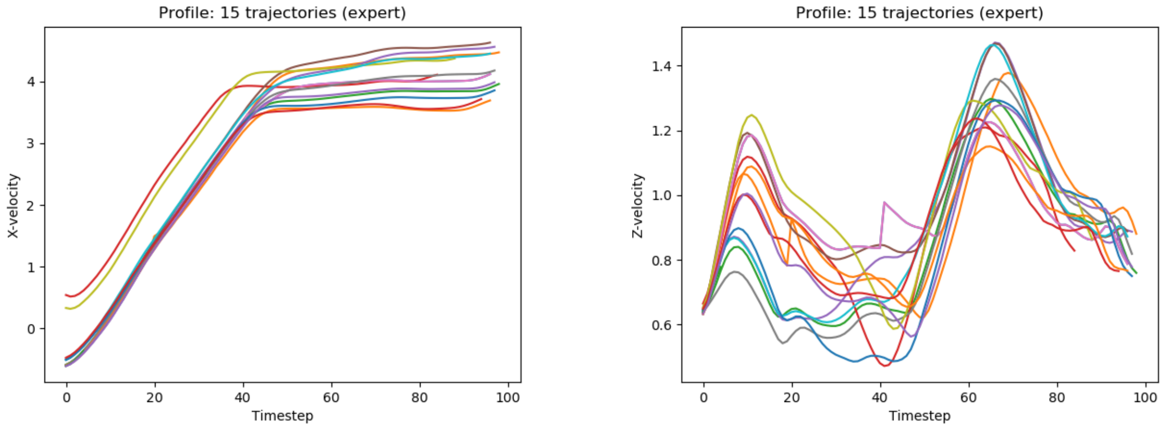
Shown below are the forward (left) and descent (right) speed profiles of the optimal human expert and the SAC-learned policy. The optimal expert performs a “hard” landing, however, the SAC-learned policy manipulates its speed during descent, resulting in smoother maneuvers.



(a) X-direction: forward.

(b) Z-direction: descent.

Figure 4.27: Optimal human expert: velocity profiles.



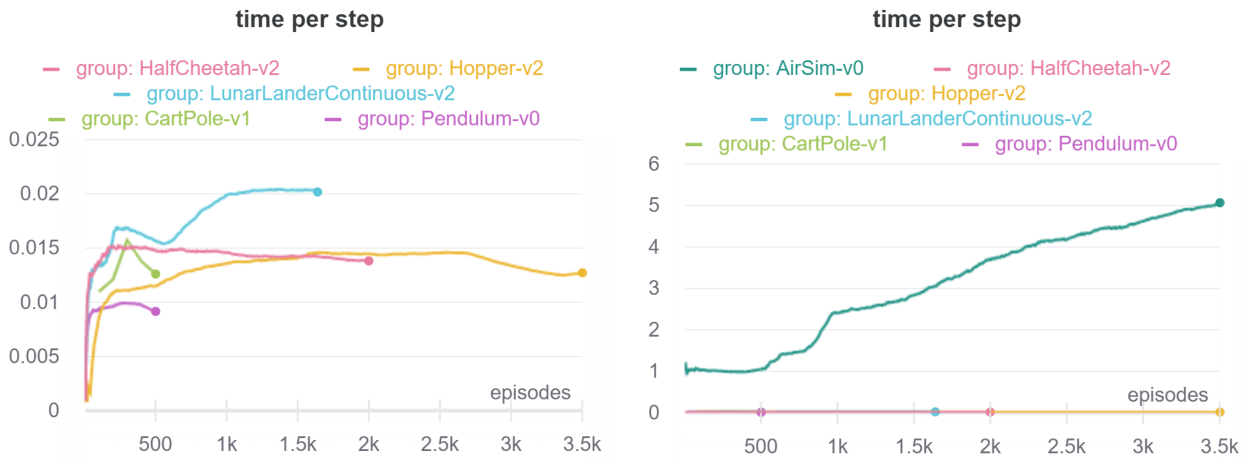
(a) X-direction: forward.

(b) Z-direction: descent.

Figure 4.28: SAC-learned policy: velocity profiles.

## 4.6 Addressing Time Bottlenecks

Figure 4.29 shows the time taken per environment interaction for training GAIL on the control tasks (Figure 4.29a) and the AirSim task (4.29b. On training for the AirSim task, the y-axis changes from milliseconds to seconds. This huge bottleneck is shown in Table 4.8, which also shows the overall training time per experiment.



(a) GAIL on control tasks.

(b) GAIL on AirSim-v0 and control tasks.

Figure 4.29: Environment samples are expensive.

| **Property** | **CartPole-v1** | **Hopper-v2** | **AirSim-v0** |
|---|---|---|---|
| Dimension (state, action) | (4, 2) | (11, 3) | (6, 3) |
| Timesteps (GAIL) | $3 \times 10^5$ | $10^6$ | $10^6$ |
| Training time | 20 minutes | 2 hours | 36 hours (at 4x) |
| Time/env interaction | 4ms | 7.2ms | 129.6ms |
| Clock speed | Processor | Processor | 4 * real-time |

Table 4.8: Time bottleneck: some numbers.

This expensive training time limits the number of experiments you can run. One feature that can indirectly help reduce runtime is by setting up multiple environments on the simulator. The concept and implementation are as shown in Figures 4.30 and 4.31 respectively.

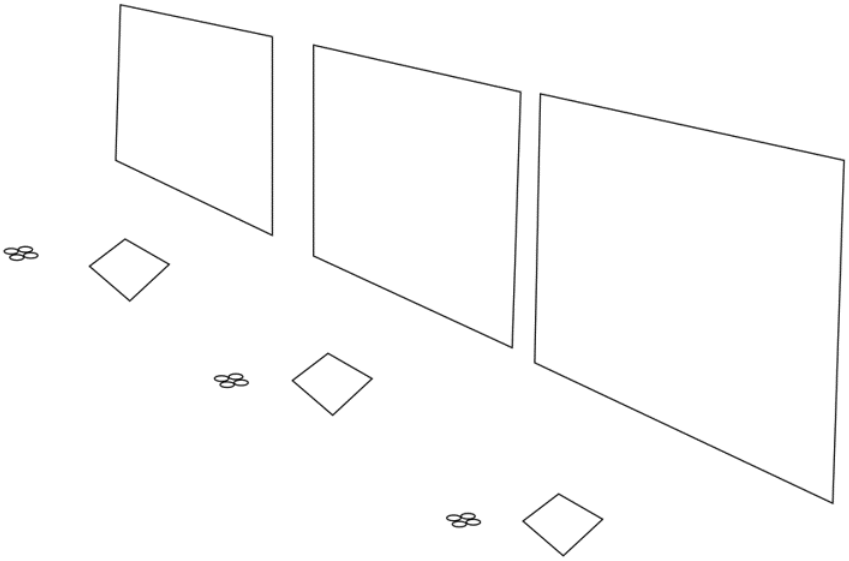


Figure 4.30: Multiple environments: concept.

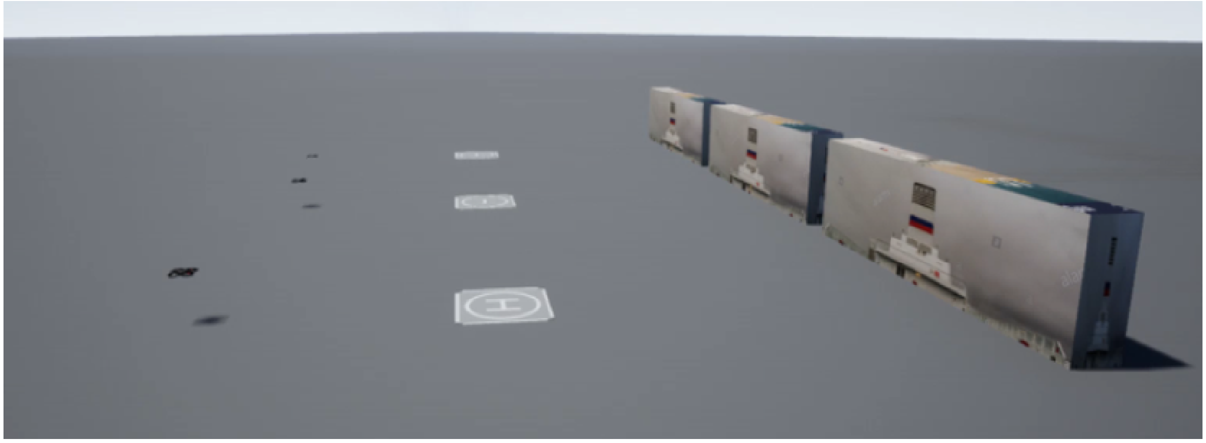


Figure 4.31: Multiple environments: implementation.

## 4.7 Observations

We list some pros and cons of GAIL in general, also observed on solving the AirSim-v0 task.

Pros:

- Can handle unknown dynamics
- Can scale to large neural network reward functions
- Can perform well on real-world tasks (with an efficient policy optimizer)

Cons:

- Adversarial optimization hard to train
- Requires smooth experts for imitation
- First-person demonstrations are typically used (no “teaching” as such)

We will now answer the questions we sought after:

1. How does imitation accuracy scale with problem dimensionality and demo data? **GAIL is sample-efficient on high-dimensional tasks** (20 trajectories)
2. How ‘smooth’ are the learned policies compared to expert policy? **smooth approach, but requires smooth experts for smooth landings**
3. Can sparse rewards be learned? At what cost? **Yes, requires larger demonstrations (>20) and careful choice of HPs**
4. Can GAIL imitate suboptimal experts? **If high-dimensional tasks do not have large variance in data**

To conclude, GAIL learns to successfully imitate the maneuvers made for approaching the landing pad on a ship deck, for both optimal and suboptimal human experts. However, GAIL requires smooth experts to accurately imitate landing. By learning a smoother policy with the help of proxy reward functions, we observed an improvement in performance over the human expert and demonstrated the benefit of smooth landings.

# 5. APPLICATION: PERFORMING TASKS ON MINECRAFT

This work was inspired by the NeurIPS 2020 MineRL Competition, a competition on sample-efficient reinforcement learning (RL) using human priors. Their goal is to promote the development of algorithms that solve hierarchical tasks with sparse rewards and long horizon.

## 5.1 Background

As we have seen earlier, current methods in Deep RL are sample-inefficient. There is also a lack of large-scale datasets for sample-efficient methods like imitation learning. This motivated the organizers to create "MineRL" [31], a large-scale dataset of solving seven tasks on Minecraft, with over 60 million (state, action) pairs.

### 5.1.1 Why Minecraft?

- Minecraft is an open-world environment, with sparse rewards and many innate task hierarchies and sub-goals
- A monthly subscriber base of 90 million users, making it easy to collect a large-scale dataset
- Microsoft Malmo, an open-source environment simulator for Minecraft, is readily available. This can also be used to collect an expert dataset

### 5.1.2 MineRL Competition 2020

The objective is to develop algorithms to mine a Diamond object in Minecraft using limited:

- Train time (4 days)
- Compute (single GPU)
- Samples from the environment simulator (8 million)

There are two competition tracks:

- **Demonstrations and Environment:** MineRL dataset and 8 million env interactions
- **Demonstrations Only:** Using only the MineRL dataset

This year's competition introduced vectorized state, action spaces that obfuscates the agent's actions. This prevents participants from using domain knowledge. We participated in the first track, hence used RL techniques combined with demonstrations for learning to perform the tasks.

Table 5.1 lists the properties of the MineRLTreeChopVectorObf-v0 Gym environment. The state consists of the point-of-view snapshots of the Minecraft simulator, and an encoded "1-D vector containing a comprehensive set of features from the game." The actions to be taken in the simulator are encoded by a "1-D vector containing keyboard presses, mouse movements (pitch, yaw), player GUI interactions, and agglomerative actions such as item crafting." In order to reduce the complexity of the action space, we represent the encoded actions by K discrete points in the action space, obtained with the help of K-means clustering. The goal is to chop 64 tree blocks.

| **Properties** | **Description** |
|---|---|
| | |
| State space (cts, dim=64*64*3 + 64) | Dict(pov, vector)<br>"pov": Box(low=0, high=255, shape=(64, 64, 3))<br>"vector": Box(low=-1.2, high=1.2, shape=(64,)) |
| Action space (cts, dim=64) | "vector": Box(low=-1.2, high=1.2, shape=(64,)) |
| Reward | 1 for each block of chopped tree |
| Termination | reward 64 or 8000 steps |
| Solved criteria | Mean reward 64 over 100 consecutive episodes |
| Expert trajectories used | [5, 10, 20] demonstrations |

Table 5.1: The MineRLTreeChopVectorObf-v0 environment.

## 5.2 Solution Approaches

The task is to chop trees in Minecraft. As one of the goals of sample-efficient RL, we want to avoid using massive datasets and hand-engineered features. This complex, hierarchical, and sparsely-rewarded task demands the use of:

- Efficient exploration techniques
- Training with human priors
- Reward shaping using IL techniques

We pick Deep Q-learning from Demonstrations (DQfD) [18], a method from the class of fD-algorithms. This can be roughly seen as Deep-Q Networks (DQN) [7], an RL algorithm, combined with demonstrations.

## 5.3 Preliminary Results

Our goal is to learn a policy that builds/improves over the expert demonstrations of a task. Specifically, train RL algorithms, paired with demonstration data, on the true reward of a task. We hope to learn the task of chopping trees and obtaining a diamond in Minecraft. For this thesis, we only look at the improvement in chopping trees on using fD algorithms: RL paired with demonstrations, compared to learning with an RL algorithm.

### 5.3.1 Tools Used

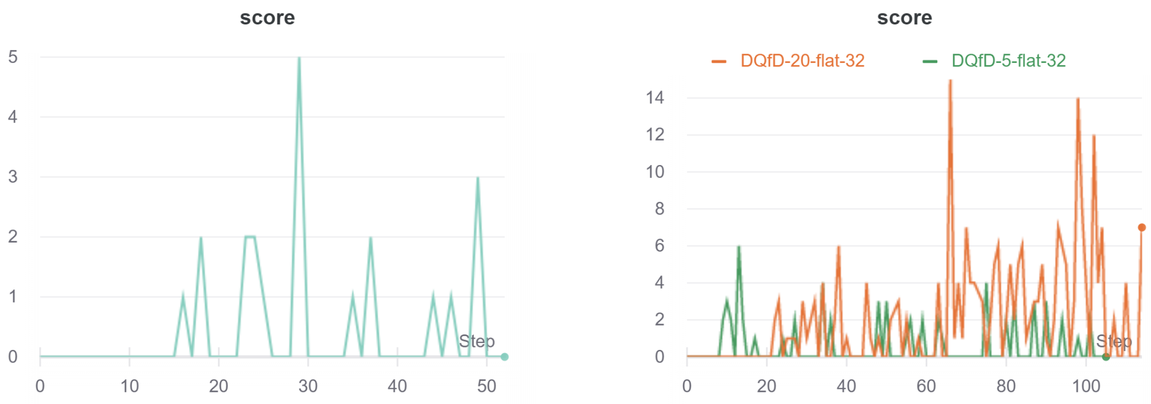
- **RL Library:** Medipixel 0.10
- **Framework:** Pytorch 1.3.1
- **Hyperparameters (HPs):** Medipixel 0.10
- **Performance metrics:** W&B 0.10

We look at the same two performance metrics as before: reward convergence and policy evaluation. The objective is to chop trees in Minecraft (MineRLTreeChopVectorObf-v0). The task is solved by chopping 64 tree blocks, where each chopped block results in a reward of 1. Table 5.2 lists the important HPs used. The remaining HPs were those of the LunarLander-v2 task on Medipixel 0.10.

| **Algorithm** | **Hyperparameters** | **Value** |
|---|---|---|
| | | |
| program | num episodes | 100 |
| | seed | 42 |
| | K-means | yes, 32 |
| common | gamma | 0.995 |
| | policy | ’C51DuelingMLP’ |
| | loss | ’C51Loss’ (RainbowDQNLoss) |
| | hidden size | [128, 64] |
| | optimizer | Adam |
| | learning rate | $10^{-4}$ |
| | epsilon decay | $5 \times 10^{-6}$ |
| | batch size | 128 |
| | buffer size | $10^5$ |
| DQfD | pre-train step | $10^3$ |
| | loss weights $\lambda_1, \lambda_2$ | 1 |
| | margin | 0.8 |

Table 5.2: Hyperparameters for Rainbow DQN and DQfD on MineRLTreeChopVectorObf-v0.

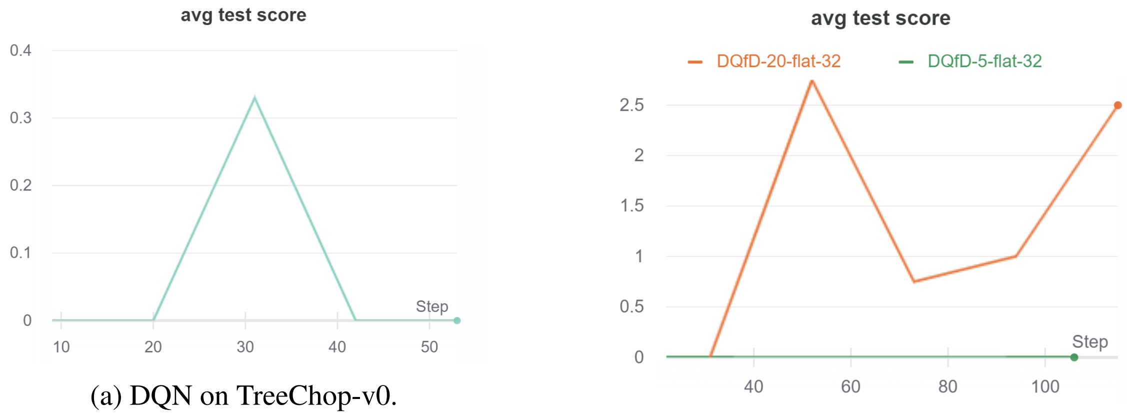
Figure 5.1 shows the performance of DQN and DQfD (roughly, DQN with demonstrations) on the task. Although neither completely solves the task, DQfD performs significantly better than DQN, improving with demonstrations. Figures 5.1a, 5.2b show the difference in reward convergence and policy evaluation (testing) respectively. While DQN’s best performance on testing was chopping a tree block (over 3 test episodes, so 0.3 on average), DQfD managed to chop 10 trees (over 4 test episodes, so 2.5 on average). This is a significant boost in performance on using 20 demonstrations, with only 100 episodes of training.



(a) DQN on TreeChop-v0.

(b) DQfD on 5, 10 demos of TreeChop-v0.

Figure 5.1: RL vs fD on MineRLTreeChopVectorObf-v0: reward convergence.



(a) DQN on TreeChop-v0.

(b) DQfD on 5, 10 demos of TreeChop-v0.

Figure 5.2: RL vs fD on MineRLTreeChopVectorObf-v0: policy evaluation.

Chopping trees in Minecraft involves a complex, extremely sparse environment that requires careful HP selection. We are working on this as part of the MineRL Competition, whose objective is to solve the MineRLObtainDiamondVectorObf-v0 environment, a task with a hierarchical structure (Figure 5.3) and exponential reward function (Figure 5.4), where the reward for obtaining the diamond is 1024. The environment is far more complex and chopping trees is just the first task.

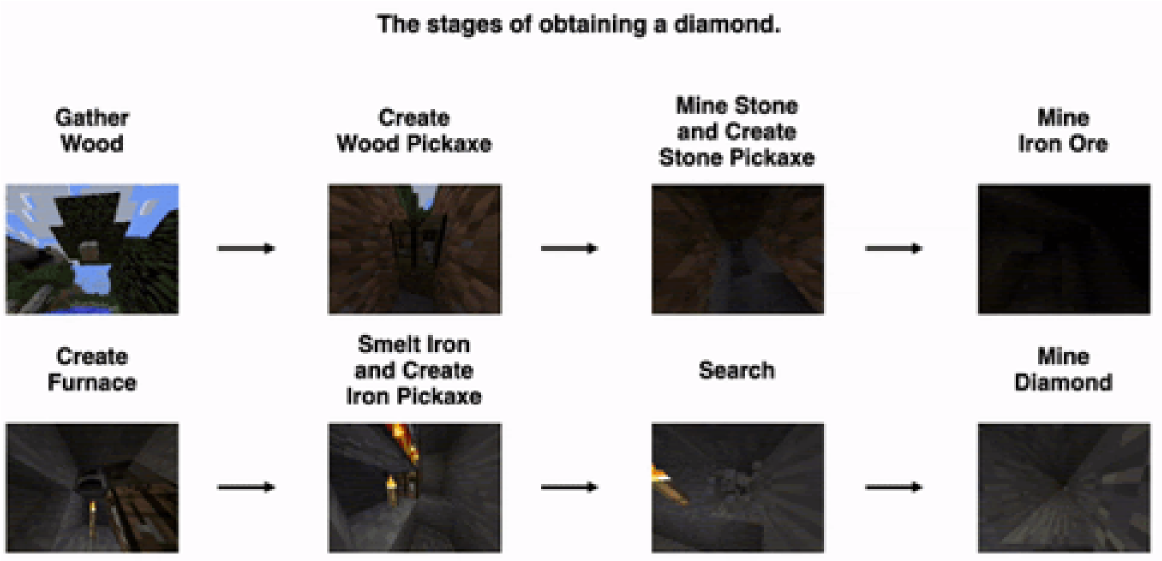

The stages of obtaining a diamond.
Gather Wood
Create Wood Pickaxe
Mine Stone and Create Stone Pickaxe
Mine Iron Ore
Create Furnace
Smelt Iron and Create Iron Pickaxe
Search
Mine Diamond

Figure 5.3: The MineRLObtainDiamondVectorObf-v0 task hierarchy. Reprinted from [4].

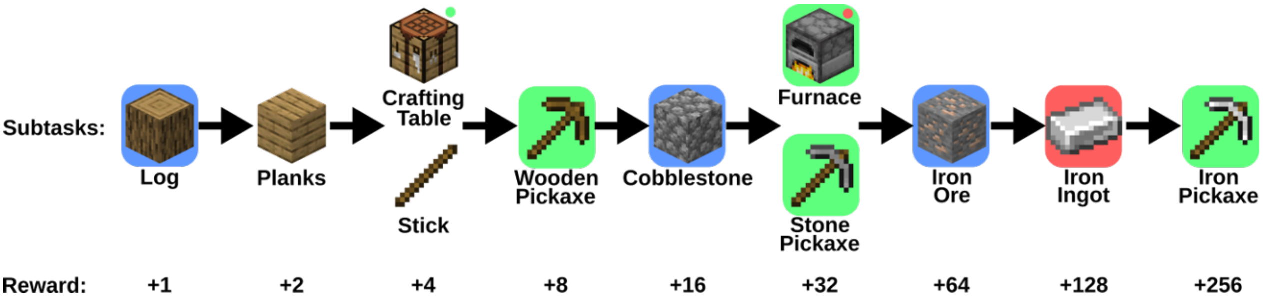


Figure 5.4: The MineRLObtainDiamondVectorObf-v0 environment reward structure. Reprinted from [4].

To summarize, we observe a boost in performance by pairing RL with demonstrations for solving the TreeChop-v0 task in Minecraft. We hope to further improve performance by exploiting the innate hierarchy of the task, briefly mentioned in Section 6. A key limitation of learning behaviors for these tasks, just like in AirSim-v0, is the time bottleneck. Training MineRLTreeChopVectorObf-v0 for 100 episodes takes roughly 2 days, which is after using human demonstrations and efficient exploration techniques. This is a challenge we hope to address with techniques like HP tuning and multiprocessing.

# 6. FUTURE WORK AND CONCLUSION

## 6.1 Future Work

From the earlier discussions and a careful look at the results, we note future directions of research and list potential improvements in algorithmic choices, simulator performance, and problem scaling for the two applications. We focused on specific algorithms for learning behavior, however, there has also been interest in some recent techniques. To mention a few, there are other adversarial learning methods [32], techniques such as multi-agent learning [33], transfer learning [34], and meta-learning [35], for learning behavior or a subset of behaviors that can be generalized to unknown environments, e.g. point-to-point maneuvers for autonomous UAVs.

### 6.1.1 AirSim: Autonomous UAV Maneuver and Landing

The bigger picture of our research is to deploy models on drones in real-time, for learning maneuvers with many degrees of freedom. For example, making approaches with yaw, landing on a moving platform, flying in the presence of wind, etc. Gazebo [36] is a better choice of simulator than AirSim for rendering these realistic behaviors, as it contains several models of quadcopters that can directly be used for real-world training. There has been recent work on learning reward functions that are robust [37] or have smoothness properties [38], which show performance improvements over GAIL on high-dimensional, sparsely-rewarded tasks.

### 6.1.2 MineRL: Solving Tasks in Minecraft

We plan to filter the image component of the observations through Convolutional Neural Networks (CNNs), to learn representations of the encoded states. Recent work shows promise in employing hierarchical learning [39, 40], meta-learning [41], and multi-agent RL methods [42] to learn both implicit and explicit hierarchies in tasks. It could also be beneficial to train on datasets of individual tasks in the hierarchy (chopping trees, navigation, survival), along with datasets for solving the complete task of obtaining a diamond. This brings in diversity in the demonstrations and can help boost performance.

## 6.2 Conclusion

This thesis was motivated by two challenges faced by modern deep Reinforcement Learning (RL) systems: (i) exponentially growing sample requirements for learning, and (ii) undesired behaviors, resulting from the explicit design of cost functions. It could be easier to learn from readily available example expert behavior, especially in tasks where environment samples and compute are expensive. To address this, we explore sample-efficient imitation learning techniques and picked Generative Adversarial imitation Learning (GAIL) for learning behaviors from expert demonstrations.

We looked at the sparsely-rewarded problem of autonomous UAV maneuver and landing (in simulation), inspired by the maneuvers pilots make to land on space-constrained ships. We designed a novel method of employing imitation learning (GAIL) to learn the pilot's behavior purely from human experts. In the finite horizon case, and with sufficient demonstrations, GAIL successfully imitates the maneuvers made by both optimal and suboptimal experts. By training on proxy rewards, RL methods learned smoother landings than human experts, necessitating the need for learning from smooth human experts.

Finally, we discussed the potential of pairing expert demonstrations with RL to solve tasks with sparse rewards, long horizon, and with restrictions on environment samples and compute. We picked the task of chopping trees in Minecraft and showed that fD-based algorithms, roughly interpreted as RL algorithms paired with demonstrations, are better at solving the task than RL methods alone.

The codebase for reproducing the experiments (except for the W&B plots) in Section 3 and Section 4 is available **here**. The README is easy to follow and the codebase includes features like Tensorboard, multiprocessing, and callbacks for performance monitoring and feedback. The codebase for Section 5 is currently private, as we are participating in the MineRL competition. It should be publicly accessible **here** on Jan 1, 2021!

# REFERENCES

[1] 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.

[2] OpenAI, "Openai five." `https://blog.openai.com/openai-five/`, 2018.

[3] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in neural information processing systems*, pp. 4565–4573, 2016.

[4] AIcrowd, "Minerl 2020 competition." `https://www.aicrowd.com/challenges/neurips-2020-minerl-challenge`, 2020.

[5] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, *et al.*, "Alphastar: Mastering the real-time strategy game starcraft ii," *DeepMind blog*, p. 2, 2019.

[6] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, pp. 1–9, 2013.

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

[8] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[9] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[10] 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," *arXiv preprint arXiv:1509.02971*, 2015.

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

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

[13] J. Clark and O. Dario Amodei, "Faulty reward functions in the wild." `https://openai.com/blog/faulty-reward-functions/`, 2016.

[14] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*, p. 1, ACM, 2004.

[15] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[16] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010.

[17] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning*, pp. 49–58, 2016.

[18] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, *et al.*, "Deep q-learning from demonstrations," *arXiv preprint arXiv:1704.03732*, 2017.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[20] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines." `https://github.com/hill-a/stable-baselines`, 2018.

[21] A. Raffin, "Rl baselines zoo." `https://github.com/araffin/rl-baselines-zoo`, 2018.

[22] L. Biewald, "Experiment tracking with weights and biases," 2020. Software available from wandb.com.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[24] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[25] A. L. Stingl, "Vtol aircraft flight system," Jan. 6 1970. US Patent 3,487,553.

[26] NATO, "Helicopter operations from ships other than aircraft carriers (hostac)," 2017.

[27] J. L. Sanchez-Lopez, J. Pestana, S. Saripalli, and P. Campoy, "An approach toward visual autonomous ship board landing of a vtol uav," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1-2, pp. 113–127, 2014.

[28] G. Xu, Y. Zhang, S. Ji, Y. Cheng, and Y. Tian, "Research on computer vision-based for uav autonomous landing on a ship," *Pattern Recognition Letters*, vol. 30, no. 6, pp. 600–605, 2009.

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

[30] Epic Games, "Unreal engine." `https://www.unrealengine.com`, 2019.

[31] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov, “Minerl: A large-scale dataset of minecraft demonstrations,” *arXiv preprint arXiv:1907.13440*, 2019.

[32] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, “Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning,” *arXiv preprint arXiv:1809.02925*, 2018.

[33] J. Song, H. Ren, D. Sadigh, and S. Ermon, “Multi-agent generative adversarial imitation learning,” in *Advances in neural information processing systems*, pp. 7461–7472, 2018.

[34] E. Parisotto, J. L. Ba, and R. Salakhutdinov, “Actor-mimic: Deep multitask and transfer reinforcement learning,” *arXiv preprint arXiv:1511.06342*, 2015.

[35] A. Zhou, E. Jang, D. Kappler, A. Herzog, M. Khansari, P. Wohlhart, Y. Bai, M. Kalakrishnan, S. Levine, and C. Finn, “Watch, try, learn: Meta-learning from demonstrations and reward,” *arXiv preprint arXiv:1906.03352*, 2019.

[36] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[37] J. Fu, K. Luo, and S. Levine, “Learning robust rewards with adversarial inverse reinforcement learning,” *arXiv preprint arXiv:1710.11248*, 2017.

[38] H. Xiao, M. Herman, J. Wagner, S. Ziesche, J. Etesami, and T. H. Linh, “Wasserstein adversarial imitation learning,” *arXiv preprint arXiv:1906.08113*, 2019.

[39] H. M. Le, N. Jiang, A. Agarwal, M. Dudík, Y. Yue, and H. Daumé III, “Hierarchical imitation and reinforcement learning,” *arXiv preprint arXiv:1803.00590*, 2018.

[40] O. Nachum, S. S. Gu, H. Lee, and S. Levine, “Data-efficient hierarchical reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 3303–3313, 2018.

[41] A. Gupta, B. Eysenbach, C. Finn, and S. Levine, “Unsupervised meta-learning for reinforcement learning,” *arXiv preprint arXiv:1806.04640*, 2018.

[42] L. Busoniu, R. Babuska, and B. De Schutter, “A comprehensive survey of multiagent reinforcement learning,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, pp. 156–172, 2008.