Self-corrective Apprenticeship Learning for Quadrotor Control

Haoran Yuan

TU Delft

Self-corrective Apprenticeship Learning for Quadrotor Control

by

Haoran Yuan

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Wednesday December 11, 2019 at 10:00 AM.

Student number:	4710118	
Project duration:	December 12, 2018 – December 11, 2019	
Thesis committee:	Dr. ir. E. van Kampen,	C&S, Daily supervisor
	Dr. Q. P. Chu,	C&S, Chairman of committee
	Dr. ir. D. Dirkx,	A&SM, External examiner
	B. Sun, MSc,	C&S

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Contents

1 Introduction

A major challenge in a flight control problem is high difficulty of obtaining an accurate model of aircraft dynamics. To avoid using a model, some model-free approaches have been developed. Some of them belong to a larger set of learning algorithm called the Reinforcement Learning (RL) [1]. RL is typically concerned with the problem about how the maximum cumulative reward can a learning agent get. In general, RL is one of the machine learning paradigms different from supervised learning as well as unsupervised learning.

Implementing Reinforcement Learning in aerospace has two major concerns. The first concern is safety. An immature policy often leads to crashes. The second concern is sample-efficiency. Sample-efficiency denotes the amount of information an agent can get from a limited amount of trials, time steps or trajectories. The sample-efficiency is a problem that many RL algorithms try to solve. In flight control, sample-efficiency is especially important because problems in this field often vary in dimensions, making it difficult for regular RL algorithms to extract useful information from experiences efficiently.

One approach to improving sample-efficiency is taking advantage of demonstration data. If a set of trajectories could be considered as an experience for a RL agent, it is natural to think other experiences could also be used to train the agent. It is expected that these experiences from a demonstrator, could provide information about how to complete the task. To incorporate these experiences, Learning from Demonstrations (LfD) was designed. The idea of LfD is to imitate the policy of the agent by extracting information from demonstrations. Usually the LfD methods can be categorized into three strategies [2],

1. *mapping* directly links actions to states,

2. *model-learning* assembles available information of sampled trajectories into reward functions or transition models,

3. *planning* selects actions based on pre- and post-conditions of states.

The first category maps actions to states based on demonstrations. This method could only succeed when providing it with a large amount of data that cover most possible agent trajectories [3]. In the second and third category, the agent does not learn to match actions to states but to explain the demonstrated behavior with reward functions or transition models.

The interest of this thesis is the model-learning LfD, more specifically the LfD algorithms that rebuild reward functions based on demonstrations. It is difficult to model the intention information for two reasons. Firstly, it is infeasible to provide all possible trajectories as demonstrations to the agent in flight control. Secondly, the demonstrator could be other controller rather than human. Learning model from demonstrations is also infeasible as it is a process that essentially involves system identifications. Therefore, the most suitable way of using demonstration data to improve sample efficiency of RL algorithms is the learning reward functions.

Apprenticeship Learning through Inverse Reinforcement Learning (AL) is one of the LfD methods that imitates the demonstrated policy by learning reward functions. But the imitation of AL is poor because reward function cannot provide the most accurate and explicit information to the agent. The AL method, similar to most LfD methods, cannot learn a policy that surpass the demonstration, in other words, the demonstration defines the upper bound [4].

The research objective is to *improve Reinforcement Learning algorithms in terms of sample-efficiency, in the context of model-free Reinforcement Learning, by evaluating and improving the state-of-the-art Learning from Demonstrations methods that are suitable for flight control.* To achieve this research objective, the following sub-goals should be achieved:

1. Select and implement state-of-the-art model-free Reinforcement Learning and Learning from Demonstrations algorithms methods and observe the difference of performance, by making an evaluation about the suitability of these methods for the selected flight control simulation.

2. Improve Learning from Demonstrations methods by making an investigation on the advantages and disadvantages of existing Learning from Demonstrations methods as well as possible solutions.

The research questions and sub-questions that need to be answered for the research objectives are:

1. Which state-of-the-art model-free Reinforcement Learning methods have the potential to be implemented in flight control?

    (a) What are the state-of-the-art model-free Reinforcement Learning methods for discrete and continuous problems?

    (b) What are the flight control problems to be solved?

2. What is the effect of incorporating the reward functions produced by existing Learning from Demonstrations methods into Reinforcement Learning algorithms?

   (a) What are the Learning from Demonstrations methods?

   (b) What are the effects of the reward functions given by Learning from Demonstrations?

3. Which Learning from Demonstrations methods is suitable for flight control?

4. How to improve Learning from Demonstrations methods?

   (a) What are the disadvantages of using the existing Learning from Demonstration methods?

   (b) What are the improvements can be made on these disadvantages?

This research includes a preliminary study of RL and LfD methods and a scientific paper about improvements that have been made on AL. In the preliminary study, basics of RL will be introduced, and some of the LfD methods that learn reward functions will be reviewed. Comparisons will be made between these methods in terms of their capability of imitating demonstrated policy and feasibility of flight control implementations. The preliminary study will answer the first three research questions then bring up the disadvantages in existing LfD methods. This research will especially focus on AL. In the scientific paper, two improvements will be made on AL. The first improvement enables the agent to learn policies from trajectories found by RL, and finally outperform demonstration's corresponding policy. The second improvement enables the agent to efficiently learn one reliable policy as a merger of all historic policies.

The remainder of this thesis is structured as follows. Part 1 contains a scientific paper that explains the Self-corrective Apprenticeship Learning (SAL) in details. Part 2 is the preliminary study of this research. Chapter 2 is a literature review that introduces basics in the RL and LfD while highlighting some state-of-the-art algorithms. Chapter 3 analyzes two specific algorithms in a corridor simulation. Chapter 4 provides a supplementary discussion on a part of SAL algorithm. Chapter 5 concludes this thesis and makes recommendations for future research.

I

Scientific Paper

Self-corrective Apprenticeship Learning for Quadrotor Control

Haoran Yuan *
Delft University of Technology, Delft, The Netherlands

The control of aircraft can be carried out by Reinforcement Learning agents; however, the difficulty of obtaining sufficient training samples often makes this approach infeasible. Demonstrations can be used to facilitate the learning process, yet algorithms such as Apprenticeship Learning generally fail to produce a policy that outperforms the demonstrator, and thus cannot efficiently generate policies. In this paper, a model-free learning algorithm with Reinforcement Learning in the loop, based on Apprenticeship Learning, is therefore proposed. This algorithm uses external measurement to improve on the initial demonstration, finally producing a policy that surpasses the demonstration. Efficiency is further improved by utilising the policies produced during the learning process. The empirical results for simulated quadrotor control show that the proposed algorithm is effective and can even learn good policies from a bad demonstration.

Nomenclature

$\mathcal{S}, \mathcal{A}, \mathcal{S}_0$	=	state space, action space, set of all initial states
s, a	=	state and action
R, P_T	=	reward function and transition probability function
γ, T	=	discount factor, time horizon
π	=	policy
τ	=	trajectory
ϕ	=	feature vector
$\mu, \bar{\mu}$	=	feature expectation and virtual feature expectation
$\mathcal{T}_D$	=	demonstrated set of trajectories
μ_D	=	feature expectation of the demonstrated set of trajectories
λ	=	relative closeness of virtual feature expectation
κ	=	relative closeness of virtual feature expectation with changing demonstrations
$\rho, \hat{\pi}$	=	distributed policy and merged policy
$\mathcal{B}$	=	trajectory buffer
$\mathcal{T}_{env}$,$\mathcal{T}^{train}$, $\mathcal{T}^{eval}$	=	set of all trajectories, trajectory set during RL training, trajectory set obtained by executing a policy
m, g	=	mass of quadrotor and gravitational acceleration
$z, \dot{z}, \ddot{z}$	=	quadrotor's vertical position, vertical velocity, vertical acceleration
T_I	=	propeller thrust in the inertial frame
v_p	=	propeller speed
d, α_{pitch}	=	diameter of propeller, propeller pitch
$\mathrm{Z}, \dot{\mathrm{Z}}$	=	discrete position and discrete velocity
$z_{target}, \mathrm{Z}_{target}$	=	control target and discrete control target
$z_{obs}, \dot{z}_{obs}$	=	states observed by PD controller
k_P, k_D	=	PD coefficients
α	=	learning rate
M_τ, M	=	performance measurement for single trajectory, performance measurement for a set of trajectories

I. Introduction

Reinforcement Learning (RL) [1] has shown the potential of exploring and acting in an unknown environment without pre-existing knowledge of environment dynamics. The learning process for RL involves the agent learning

*Master student, Control & Simulation, Aerospace Engineering, Delft University of Technology, Delft, The Netherlands

the optimal policy to maximise the expected sum of a discounted reward over time through engaging in agent-environment interactions [1]. In a standard Markov Decision Process (MDP), the agent receives a specific reward signal when visiting a state by means of an action. In value-based RL, such reward signals are usually translated into values of states or state-action combinations, causing the optimal policy to be to find the action that leads to a state with a higher state value.

In aircraft control, dynamics are often expensive to fully identify. RL thus provides a model-free learning framework capable of reducing the effect of unknown dynamics in aircraft control. However, the cost of using RL is the large number of sampled agent-environment interactions required to avoid sub-optimal policy development where the agent must choose between exploitation and exploration. The former enables the agent to use a policy that is optimal based on the current knowledge, while the later forces the agent to discover other possible solutions[1]. For real aircraft, a large number of interactions is usually infeasible due to safety concerns, however, and thus aircraft control with RL requires more sample-efficient methods.

The reward function reflects the intention of the human designer for an RL algorithm for aircraft control. In many ways, a reward function affects the sample efficiency; providing the most attractive rewards in the target state and states that could transit to the target state, or punishing the states that lead to undesirable performance, are common. Some control tasks are complex, and sample-efficient reward functions are difficult to handcraft, however. Inverse Reinforcement Learning (IRL) [2] generates a suitable reward function from a set of demonstrated trajectories, and this generated reward function can express the purpose of the demonstrator. However, generating an IRL algorithm requires full knowledge of the transition function of all relevant dynamics. To counter this, Apprenticeship Learning through Inverse Reinforcement Learning [3] uses the trajectories experienced by the agent to determine the similarity between the policy represented by demonstrated trajectories and the policies used by the agent. The behaviour of demonstrated trajectories is then characterised by a set of features in which the reward function is linear.

Apprenticeship Learning measure the similarity of policies, based on their feature expectations, to the demonstrated policies to compute a reward function; each iteration contains an RL learning process that learns from the reward function and then evaluates the similarity of the produced policy and the demonstrated policy.

The core of Apprenticeship Learning (AL) is to imitate a policy by matching the expected value of its feature expectation, calculated based on the feature expectations of its trajectories. However, this generates several problems. No policy information is included in feature expectation of each trajectory; instead, it is included in the distribution of the trajectories. Hence, many feature vectors could eventually lead to the same policy, while different policies could have the same feature expectations [4]. This problem can be alleviated using the principle of Maximum Entropy [4–6], which limits trajectories that have the same value to having the same distribution probability for a given policy; this probability grows exponentially with the value of the trajectory [4]. As the distribution of trajectories can be built more accurately when more data are collected, using the collected trajectories during training to refine the distribution model [7, 8] can also reduce the ambiguity of feature expectation. In addition, the final reward function has to be trained using an RL agent to obtain a policy for the control task, while all the intermediate reward functions and policies are discarded; the final reward function cannot reliably provide good policy, however, as the performance of the AL depends heavily on the quality of the demonstrated trajectories. AL thus learns both good and bad policies from demonstrated policy [9]. To surpass the performance of the demonstration, game theory can be applied to the AL [9] to generate a zero-sum game between an agent that maximises the expected sum of returns by adjusting the policy π and an agent that minimises this value by tuning the coefficient of the reward function. This game-theoretic AL utilises a post-processing procedure to reduce the difference in feature expectation between the learned policy and the demonstrated one. When applied to flight control, system identification can thus be used to surpass the performance of demonstrations [10]. As the demonstrations for flight control are typically provided by humans, and the demonstrated trajectories may thus contain noise, a system identification process for dynamics related to trajectories in demonstrations can also help reduce this noise.

To conclude, the selection of feature vector and the way to calculating the feature expectations greatly affect the performance of AL. However, AL still always learns similar policies to the demonstrated one. AL for flight control can adopt the approach of system identification to reduce the effect of underperforming trajectories but this approach is only suitable for the situation when the undeerperforming trajectories are caused by the noise. The algorithm that learns different but better policy rather than the demonstrated one is not yet available.

The contribution of this paper is the development of Self-corrective Apprenticeship Learning (SAL), a model-free learning scheme based on both AL and RL. SAL consistently searches for policies that outperform the demonstration by merging all past policies into a more reliable one. SAL thus offers two improvements on the original AL algorithm. The first is a modified projection method that reduces the effect of the environment when extracting the general policy

information from the demonstrated trajectories. The second is the introduction of a trajectory evaluation and replacement mechanism. Trajectories inside the initial demonstration are assumed to underperform; thus, by introducing an external performance measurement, trajectories with better performance will replace the trajectories from the demonstration.

The remainder of this paper is structured as follows. Section II introduces the basics of RL and AL, while in Section III, a brief review of the projection method used in the original AL algorithm is given, followed by the modified projection method, the Policy Fusion Projection. In Section III, a trajectory evaluation and replacement mechanism is also introduced, and the full SAL algorithm is outlined. In Section IV and Section V, the experiment setups and results are presented and discussed. Finally, Section VI concludes this paper.

II. Preliminaries

A Markov Decision Process (MDP) can be formulated in a tuple $(\mathcal{S}, \mathcal{A}, P_T, R, \mathcal{S}_0, \mathcal{T}_{env}, \gamma, T)$. The state space in such an MDP is represented by the set $\mathcal{S}$, which defines all possible states. Similarly the action space $\mathcal{A}$ includes all actions in MDP. P_T is the transition probability function of the environment, and the probability of transiting into the state s_t with s_{t-1} and a_{t-1} can be written as $P_T(s_t|s_{t-1}, a_{t-1})$, where footnotes t and $t-1$ show time steps. The reward function is R where $R(s_t, a_t)$ is the reward received by the agent when selecting action a_t at state s_t. $\mathcal{S}_0$ is the set of initial states. The initial state is defined as the first visited state of each trajectory. All possible trajectories in this MPD are summarised by the set $\mathcal{T}_{env}$. γ is the discount factor for rewards and T is the time horizon of trajectory.

A trajectory τ is a connection of states and actions selected in these states; thus, $\tau = \{(s_0, a_0), (s_1, a_1), ..., (s_T, a_T)\}$. A policy π is the mapping from a state to the probability of action, with $\pi(s, a) = P(a|s, \pi)$. A set of trajectories can be obtained by enacting a given policy. Given the set of initial state $\mathcal{S}_0$, a set of trajectories related to a policy can be calculated. The AL thus assumes that if two sets of trajectories overlap, the executed policies are the same [4]. In AL, feature expectation is used to measure such similarity between sets of trajectories. The feature expectation of policy π is denoted by $\mu(\pi)$, and the expression of $\mu(\pi)$ is

$$\mu(\pi) = \sum_{\tau \in \mathcal{T}_{env}} P(s_0) \prod_{t=1}^{T} \pi(s_t, a_t) P_T(s_{t+1}|s_t, a_t, \tau) \sum_{t=0}^{T} \gamma^t \, \phi(s_t, a_t) = \sum_{\tau \in \mathcal{T}_{env}} P(\tau|\pi) \sum_{t=0}^{T} \gamma^t \, \phi(s_t, a_t) \tag{1}$$

where $P(s_0)$ is the probability of initial state s_0.

The feature expectation of a policy can be estimated by the feature expectation of the trajectory set produced by that policy: $\mu(\pi) = E[\mu(\mathcal{T}, \pi)]$. For convenience of discussion, this paper assumes that feature expectation of the set of trajectories equals the feature expectation of the executed policy. Formally, the feature expectation μ of the set $\mathcal{T}$ is $\mu(\mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \sum_{t=0}^{T} \gamma^t \, \phi(s_t, a_t)$, where $\phi(s_t, a_t)$ is the column feature vector of state s_t and action a_t in trajectory τ_i at time step t. Notation $|\mathcal{T}|$ refers to the number of elements in set $\mathcal{T}$. The reward function in AL is thus the linear combination of features in vector ϕ, and the reward function is $R(s, a) = w^\intercal \cdot \phi(s, a)$ where w is the vector of weights of features.

In the AL, a demonstration is provided as a set of trajectories obtained by executing the demonstrator's policy. The demonstrated set of trajectories is thus denoted by $\mathcal{T}_D$. The AL algorithm measures the feature expectation μ_D of the demonstration, then calculates coefficient w for reward function R n order to gradually improve the similarity between the imitating policy and the demonstrated policy. In general, the methods used to calculate w can be divided into two categories: the first includes generic quadratic programming methods such as Support Vector Machines, while other features the projection method [3]. Further detail about the projection method is discussed in the next section.

III. Projection Method

A. Review of the Projection Method

In AL, any generic quadratic programming algorithm can be used to compute the coefficient w. The projection method computes the coefficient w progressively, however, ensuring an improvement in feature expectation at every iteration. The AL algorithm produced with the projection method is shown in Algorithm 1. As seen in Figure 1, the projection can be illustrated as the process of finding the closest point in the feature expectation space to the feature expectation of the demonstration.

The AL algorithm poduced using the projection method does not utilise all of the information from all policies trained during the process; instead, only the policy trained in the last iteration, with its respective reward function, is

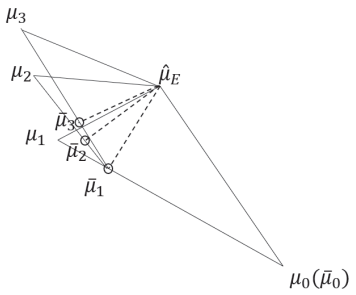


Fig. 1 Illustration of the projection method, adapted from [3].

Algorithm 1 Apprenticeship Learning using the Projection Method, adapted from [3]

Input: Demonstration $\mathcal{T}_\mathcal{D}$, a RL algorithm, stopping criteria e.
Randomly initialize a policy and estimate its feature expectation μ_0
Set $\bar{\mu}_0 = \mu_0$
Estimate the feature expectation μ_D of demonstration
$i = 0$
while $\|\mu_D - \mu_0\| > e$ **do**
  $i = i + 1$
  Calculate coefficient w with $w = \mu_D - \bar{\mu}_i$
  Compute the optimal policy π_i with the RL algorithm when the reward function is $R = w^\intercal \cdot \phi(s_t, a_t)$
  Estimate the feature expectation μ_i of policy π_i
  Compute λ_i with $\lambda_i = \frac{(\mu_i - \bar{\mu}_{i-1})(\mu_D - \bar{\mu}_{i-1})}{(\mu_i - \bar{\mu}_{i-1})(\mu_i - \bar{\mu}_{i-1})}$
  Compute $\bar{\mu}_i$ with $\bar{\mu}_i = \lambda_i \mu_i + (1 - \lambda_i)\bar{\mu}_{i-1}$
end while
Output: reward function $R = w^\intercal \cdot \phi(s_t, a_t)$ and an optimal policy corresponding to this reward function.

used. This leads to a waste of samples. It can be shown that the projection method guarantees approximating virtual feature expectation, as $\bar{\mu}_i$ moves closer to μ_D with each iteration [3]. However, μ_i has no such property, and it is more often the case that μ_i revolves around μ_D at a fixed distance.

In AL, in order to obtain the policy best related to feature expectation $\bar{\mu}_i$, policies are computed by RL during all iterations in order to construct a *distributed policy* using a post-processing procedure [9]. In Algorithm 1, λ_i represents the possibility of each policy occurring; the more similar the policy is to the prediction, the larger λ_i will be. Let ρ_i be the distributed policy with feature expectation $\bar{\mu}_i$; if ρ_i is a distribution over all past policies in AL, the distributed policy ρ_i can thus be written as

$$\rho_i = \begin{cases} \pi_i, & p_i = \lambda_i \\ \pi_{i-1}, & p_{i-1} = (1 - \lambda_i)\lambda_{i-1} \\ \vdots & \vdots \\ \pi_0, & p_0 = (1 - \lambda_i)(1 - \lambda_{i-1})...(1 - \lambda_1) \end{cases} \tag{2}$$

where p_i is the probability of choosing the corresponding policy under the distributed policy ρ_i.

For aircraft control, the use of distributed policy ρ_i may cause safety and reliability concerns, as in most early iterations, and some later iterations, AL will produce policies with poor performance despite the demonstrator performing well; this occurs because the distributed policy ρ_i has a possibility of using any past policy, with no regard to its performance nor to similarities with the demonstrator's policy. This occurs because ρ_i is not in itself a policy but rather a distribution of all policies produced during AL. Moreover, if the demonstrator does not perform well, the combined policy will learn a policy with poor performance. The demonstration thus defines the upper bound of agent performance [9].

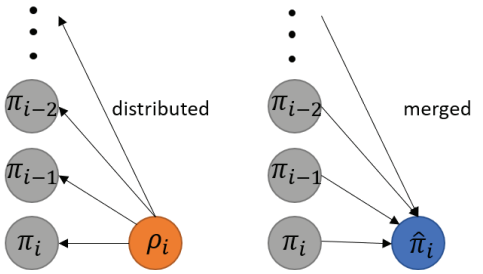


Fig. 2 Left: ρ_i is the distribution of all historic policies from AL. Right: $\hat{\pi}$ is a merger of all historic policies.

B. Policy Fusion Projection Method

To incorporate all the policies developed during the AL's training process, the Policy Fusion Projection (PFP) method assigns weights to all historic policies before selecting the most likely course of action. Unlike the combination of policies discussed in the previous section, the Policy Fusion Projection Method can produce both deterministic policies, though these policies are again independent of historic policies.

$$\kappa_i = \frac{(\mu_i - \bar{\mu}_{i-1})(\mu_D - \bar{\mu}_{i-1})}{(\mu_i - \bar{\mu}_{i-1})(\mu_i - \bar{\mu}_{i-1})} \tag{3}$$

In the PFP method, κ_i is used as a weight for voting; κ_i can be calculated by using equation 3. Note that although κ_i has the same expression as the λ_i in Algorithm 1, the feature expectation μ_D in κ_i is not necessarily fixed as it is for λ_i. Every policy generated by RL is assigned a κ. Let $\hat{\pi}_i$ be the policy computed using the PFP method, and let $\hat{\mu}_i$ be the feature expectation of this policy; policy $\hat{\pi}_i$ can thus be written as

$$\hat{\pi}_i \;=\; \kappa_i \pi_i \;+\; (1 - \kappa_i)\hat{\pi}_{i-1} \tag{4}$$

where $\kappa_i = \lambda_i$ in the AL algorithm. It will be shown later that, in the proposed algorithm, κ_i is not equal to λ. A comparison between the original projection method and the PFP method is illustrated in Figure 2

The PFP method introduces a majority voting mechanism for all historic policies. Applying the equation 4 enables the agent to choose one policy to follow for each state according to the weighted sum of votes assigned given by historic policies rather than blindly opting for one policy at each time step in the trajectory. When the agent must choose an action, it evaluates the desirability of each action by summing up these votes, multiplied by their weights κ; the higher the resulting total, the more likely the action is to be selected.

IV. Self-corrective Apprenticeship Learning

A. Trajectory Evaluation and Replacement

In RL, the agent learns to finish a task by evaluating all states and actions. A handcrafted reward function is thus usually used to help the agent understand the task. In AL, the trajectories in the demonstration are assumed to be near-optimal for completing the task [11]. If this assumption does not hold, the agent will learn from trajectories with poor performance, and the performance of the agent will also be poor. In AL, each trajectory is evaluated based on its feature expectation, and all feature expectations of all trajectories together form the feature expectation of the policy producing those trajectories. However, neither the feature expectation of trajectories nor that of policy contains information about how well the task is accomplished. The agent thus follows the demonstration and learns to apply a similar policy while having no knowledge about the task. External measurement of the trajectory can thus help the agent evaluate different trajectories of demonstrations.

Choosing a suitable external performance measurement is a non-trivial task. Evaluating the trajectories requires realistic expectations of outcomes that are observable. For example, if the agent is expected to reach at certain target states, the trajectory evaluation should involve a measurement of whether or not the target states have been reached. In RL the performance of trajectory can be measured by trajectory's value which is the accumulative state values in the trajectory [3]; this is based on the fact that the reward function has correctly assigned high rewards to target states. In tracking control this reward function can be decreasing from the highest reward at the target state to the lowest

reward at furthest states from the target state. When the correct reward function is not available, the suitable choice of the performance measurement of trajectory still depends on the expected outcomes. For example, in tracking control the suitable measure of trajectory can be the sum of tracking error over the entire trajectory. Therefore any external performance measurement of trajectory is suitable when it measures the difference between the expected outcomes and the observed ones.

Assuming that an external performance measurement of trajectory is available, the demonstrated trajectories can be replaced by trajectories with better performance measurements during the learning process. As AL includes an RL algorithm, this trajectory search can be done by RL. The learning process of AL with changes to demonstrated trajectories is thus composed of two parts: one is the RL algorithm, which searches trajectories while computing the policy based on the generated reward function, and the other is the projection method, which tries to generate a reward function that enables the RL to learn a similar policy to the policy of the demonstration.

AL with trajectory replacement is analogous to Monte Carlo RL [1] and Deep Q Learning [12]. Monte Carlo RL typically evaluates and updates state values after experiencing the entire trajectory, selecting the policy that is optimal for these state values. AL directly learns the policy of the trajectory, and each trajectory is evaluated as a whole. Hence, the external performance measurement for trajectories evaluates the whole trajectory for every RL process. Deep Q Learning introduces an experience replay mechanism to reduce the variance of neural networks [12]: demonstration trajectories can be put in the replay buffer in order to reduce the learning time [13]. With trajectory replacement, AL learns from this buffer of trajectories.

Generic AL has been modified in two aspects for this work. The PFP method modifies the projection method to prevent the agent from pursuing bad policies in a given trajectory. The trajectory evaluation and replacement mechanism also improves the quality of demonstration in terms of performance measurement. Self-corrective Apprenticeship Learning (SAL) is an algorithm based on AL with both of these modifications. SAL learns a policy that imitates the policy of the demonstration while still changing trajectory when any trajectory with better external performance measurements is found. The full algorithm for SAL is thus presented and discussed in the next section.

B. Algorithm for Self-corrective Apprenticeship Learning

The algorithm for SAL is 2. The inputs are set using $\mathcal{T}_D$, an RL algorithm, stopping criteria e, and a performance measurement M_τ for each trajectory. The output of the algorithm is a merged policy $\hat{\pi}_i$. To ensure only useful information is placed in the demonstration buffer, trajectories with poor performance must be discarded. Therefore, performance measurement M_τ is needed, and the size of the demonstration buffer is set to 1, keeping only the most useful trajectory. This size indicates that, for each initial state, there can be only one trajectory in the demonstration buffer at a time. As the initial state of each trajectory may influence the performance measurement, trajectories with different initial states must be measured independently. The output of SAL is thus a stochastic merged policy; however, for the quadrotor control task, the policy can be made deterministic to reduce variance by choosing the most probable action in each state.

To obtain more trajectories that may outperform the policy in the demonstration, policies in RL are executed in a stochastic manner: agents have a probability, denoted by ϵ, of not acting according to the policy. This stochastic policy execution boosts the ability of SAL to find potentially better policies.

SAL is an AL algorithm augmented by external rewards. The performance measurement serves as an extra source of information for the agent, allowing it to regulate its behaviour based on external performance measurements. Thus, the performance measurement plays the same role as the reward function in RL. SAL resembles the Monte Carlo RL [1], as in Monte Carlo RL the update is also done after finishing each trajectory. However the Monte Carlo RL usually learns policies based on value functions, while SAL learns policies by matching feature expectations. SAL can thus be seen as a method that combines AL and RL.

V. Experiment Setup

A. Quadrotor Simulation

To validate SAL, a quadrotor simulation environment was built. The dynamics of the quadrotor in this environment were adapted from the quadrotor model in [14]. This environment simulates a quadrotor with one degree of freedom and one control input. The controller aims to complete the vertical position tracking task by directly changing the propeller speed. The dynamic thrust equation is adopted from [15], and the quadrotor can fly vertically with different velocities;

Algorithm 2 Self-corrective Apprenticeship Learning

Input: The demonstrated set of trajectories $\mathcal{T}_D$, an RL algorithm, stopping criteria e, performance measurement M_τ for single trajectory.
Initialize an empty buffer $\mathcal{B}$ to store trajectories.
Execute a random policy $\hat{\pi}_0$, obtain trajectory set $\mathcal{T}_0$
$\mu_0 = \frac{1}{|\mathcal{T}_0|} \sum_{\tau \in \mathcal{T}_0} \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t)$
$\bar{\mu}_0 = \mu_0$
for s_0 in $\mathcal{S}_0$ **do**
  Evaluate every trajectory in $\mathcal{T}_0$ that starts with s_0, using measurement M_τ.
  Store the trajectory with the best M_τ in buffer $\mathcal{B}$.
end for
$\mu_{D(0)} = \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \sum_{t=0}^{T} (\gamma^t \phi(s_t, a_t))$
$i = 0$
while $\|\mu_{D(i)} - \mu_0\| > e$ **do**
  $i = i + 1$
  $w = \mu_D - \bar{\mu}_i$
  Compute the optimal policy π_i with the RL algorithm when the reward function is $R = w^\intercal \cdot \phi(s_t, a_t)$, and store all trajectories during training as $\mathcal{T}_i^{train}$.
  Use *stochastic policy execution* to execute π_i, obtain trajectory set $\mathcal{T}_i^{eval}$
  $\mu_i = \frac{1}{|\mathcal{T}_i^{eval}|} \sum_{\tau \in \mathcal{T}_i^{eval}} \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t)$
  $\mathcal{T}_i = \mathcal{T}_i^{train} \cup \mathcal{T}_i^{eval}$
  Evaluate each trajectory in D_i using the performance measurement.
  for s_0 in $\mathcal{S}_0$ **do**
    Evaluate every trajectory in $\mathcal{T}_i$ that starts with s_0, using measurement M_τ.
    Store the trajectory with the best M_τ in buffer $\mathcal{B}$.
  end for
  $\mu_{D(i)} = \frac{1}{|\mathcal{B}|} \sum_{\tau \in \mathcal{B}} \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t)$
  for $j = 1, 2, ..., i$ **do**
    $\kappa_i = \frac{(\mu_j - \bar{\mu}_{j-1})(\mu_{D(i)} - \bar{\mu}_{j-1})}{(\mu_j - \bar{\mu}_{j-1})(\mu_j - \bar{\mu}_{j-1})}$
    $\bar{\mu}_j = \kappa_j \, \mu_j + (1 - \kappa_j) \bar{\mu}_{j-1}$
    Estimate $\hat{\pi}_j$ with the PFP:
      $\hat{\pi}_j = \kappa_i \pi_j + (1 - \kappa_j) \hat{\pi}_{j-1}$
  end for
end while
Output: Policy $\hat{\pi}_i$

hence, the equation of motion is

$$m\ddot{z} = -mg + T_I \tag{5}$$

where m is the mass of the quadrotor, g is the gravitational acceleration, $\ddot{z}$ is the vertical acceleration in the inertial frame, and T_I is the propeller thrust in the inertial frame. As the motion of the quadrotor is restricted, the thrust T_I equals the vertical thrust in the body frame. The thrust T_I can be summarised as

$$T_I = 4.392399 \times 10^{-8} \cdot v_p \frac{d^{3.5}}{\sqrt{\alpha_{pitch}}} (4.23333 \times 10^{-4} \cdot v_p \cdot \alpha_{pitch}) \tag{6}$$

where v_p is the rotational velocity of the propeller, d is the diameter of the propeller, and α_{pitch} is the propeller pitch. In the simulation, the four propellers have the same parameters, though two of the four propellers have a fixed rotational velocity, equal to 5,300 RPM, while the other two change simultaneously depending on the controller's choices. The values of these parameters are shown in table 1. The state space is two-dimensional based on the vertical position z and vertical velocity $\dot{z}$, while the action space has only one dimension, the propeller speed of the controllable propellers.

To further simplify this environment for RL and AL, boundaries of position and speed are used to limit the state

Table 1 Parameters for the quadrotor simulation

Parameter and unit	value
m (kg)	1.2
g (m·s^{-2})	9.81
v_p (RPM)	[5200, 5500]
d (in.)	10
α_{pitch} (in.)	4.5

space. The continuous states and control inputs are divided into 231 discrete states and 2 control inputs. The set $\mathcal{S}_0$ in this simulation thus contains 231 initial states. The position is discrete for 21 states, ranging from 0 to 20, and the velocity is discrete for 11 states, ranging from 0 to 10. Each discrete position represents a 0.29m margin of real position, and each discrete velocity represents a 0.09m·s^{-1} real velocity margin. The setting of discrete position and velocity is determined by the maneuverability of quadrotor. 21 states of discrete position ensure that the quadrotor has enough vertical space to show the performance of controller while maintaining a reasonable space for each discrete state. If the real position margin represented by each discrete position is too large, the agent would not be able to correctly perceive its current state, and thus discrete states would not be representative. If the real position margin is too small, the total number of states would become too large, the increased amount of required computational resources would make the experiment infeasible. The number of discrete velocity is determined by the same reason. 11 states are representative for the agent to perceive and are not too many to be effectively computed.

The control task must thus track the central position, implying that the target discrete state is the state $[10, 5]$. There are only two choices for discrete action, 0 or 1: 0 gives the quadrotor downwards acceleration with a propeller speed of 5,275 RPM, while 1 accelerates the quadrotor upwards with a propeller speed of 5,425 RPM. The quadrotor has a maximum speed. If it travels across the boundary of position, it is reset to a random initial state and receives a -100 reward. The simulation also uses a finite time horizon, with each trajectory containing 200 time steps; each time step equals 0.2 seconds of simulation time. The settings for this experiment are summarised in Table 2.

Table 2 Settings for the quadrotor simulation

Name and unit	value
Range of continuous z (m)	$[-3,\ 3]$
Range of continuous $\dot{z}$ (m·s^{-1})	$[-0.5, 0.5]$
Simulation time for each time step (s)	0.2
Trajectory length (time steps)	200
Name of discrete position Z	$0, 1, 2, ..., 20$
Name of discrete velocity $\dot{\text{Z}}$	$0, 1, 2, ..., 10$
Total discrete states	231
Total initial states	231
Name of discrete actions	0 and 1
Total discrete actions	2
Name of target discrete state	discrete z: 10, discrete $\dot{z}$: 5

B. Demonstrations

Demonstrations were provided using a PD controller, and the states observed were discretised. PDs with different coefficients provided demonstrations with various performances, particularly where PD was underdamped or overdamped. Obtaining demonstrations from a PD controller maintained stable demonstration performance, allowing comparisons between the performance of quadrotors trained using different methods and different demonstrations to be made. The

control of the quadrotor can be summarised as

$$v_{target} = k_P(z_{target} - z_{obs}) + k_D(0 - \dot{z}_{obs}) \tag{7}$$

where k_P and k_D are the parameters under PD control; z_{target} is the position that must be tracked, with the tracking target set to be the center of the positional range; and z_{target} is 0. z_{obs} and $\dot{z}_{obs}$ are the observed vertical position and velocity at each step. The observed position and velocity are regenerated from the index of discrete states. The purpose of using z_{obs} and $\dot{z}_{obs}$ is to ensure that RL and AL with discrete states and actions are capable of fully expressing the intention of demonstration where the PD control receives more information about states and actions. v_{target} is the target propeller speed; however, the real propeller speed was held between 5,275 RPM and 5,425 RPM depending on which one was closer to the target propeller speed. The settings for the PD control are shown in Table 3.

Table 3 Coefficients for the PD control

	k_P	k_D
overdamped	70000	70000
oscillated	70000	0

Each demonstration consisted of 2,000 trajectories, covering all initial states in $\mathcal{S}_0$. For simplification, the demonstration produced by the overdamped PD is referred to as the good demonstration and the demonstration produced by the oscillated PD is referred to as the bad demonstration.

C. Reinforcement Learning

The RL algorithm is used for comparisons; a part of both AL and SAL is tabular Q-learning. Q-learning estimates the optimal policy based on action values that are constantly updated. According to the literature [1], the discount factor is usually selected within the interval $[0.9, 1)$; in this case, the discount factor γ was set to 0.9. For Q-learning, the learning rate α is often set to 0.1 for all time steps; however, while for deterministic problems, a constant α is sufficient for convergence, for stochastic problems, such as quadrotor control with discrete states as seen in this paper, a learning rate that gradually decays to 0 facilitates the convergence of the algorithm[1]. Hence, in this case, the learning rate α was decayed from 0.2 to 10^{-4} over the time steps shown in equation 8.

$$\alpha = \max(10^{-4},\ 0.2 \times 0.99999^{t-1}) \tag{8}$$

The reward function for the comparable RL method is the negative value of the tracking error. The further the quadrotor is from the expected value, the less the agent is rewarded. The reason for using this reward function is that it measures the value of states similarly to the measurement of trajectory used in the SAL algorithm. The similarity of the ways states and trajectories are evaluated makes the comparison between the policy trained by the Q-learning with the described reward function and the policy trained by SAL with the described performance measurement more valuable.

When comparing RL with handcrafted rewards for AL or SAL, it is fairest to compare the policies produced when the same amount of samples are used to train the agents. The training of RL was thus divided into phases, and the total number of phases set to equal the total number of iterations for AL or SAL. Each training phase also contained the same number of samples as seen in one iteration of AL or SAL. For example, one iteration in SAL contains 1,000 trajectories for training RL, and 231 trajectories for validating the RL policy; hence 1,231 trajectories are used in one iteration, and one training process of RL must also contain 1,231 trajectories.

D. Setups for AL and SAL

The performance of a policy can be measured by examining the corresponding set of trajectories. The performance of the set of trajectories can be measured as the average sum of the absolute value of tracking error of each trajectory. The lower the average sum of tracking error, the better the performance. This measurement can be written as

$$M(\mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \sum_{t=0}^{T} |\mathrm{Z}_{target} - \mathrm{Z}_t| \tag{9}$$

where $\mathcal{T}$ is the set of trajectories that needs to be measured; Z_{target} is the target discrete position, which is 10 in this case; and Z_t is the discrete position in trajectory τ at time t, ranging from 0 to 20. However, Algorithm 2 requires a measurement of performance for each single trajectory. As with the measurement of sets of trajectories, the measurement for single trajectories can be done by summing the absolute value of tracking errors. This measurement can thus be written as

$$M_\tau(\tau) = \sum_{t=0}^{T} |Z_{target} - Z_t| \tag{10}$$

As Algorithm 2 shows, the performance measurement determines which trajectory experienced by the agent is stored in the demonstration buffer.

The AL algorithm uses both the original projection method and the PFP method to produce policies, while SAL uses only the PFP method, as described in Algorithm 2. The feature vector for both AL and SAL algorithms is the thus re-arranged state. Using the discrete states and actions described in Section V.A, the total number of state-action combinations is 462 including 231 state combinations and 2 actions. Thus the feature vector for AL and SAL is a 462-dimensional vector.

VI. Experiment Results

This section offers a comparison of policies developed by RL, the original AL, AL with PFP method, and SAL. Two demonstrations are provided for each of these algorithms, one based on the *good demonstration*, given by an overdamped PD, the other based on the *bad demonstration*, given by a PD with harmonic oscillation. The RL policies trained with handcrafted rewards are also be shown in all figures as a benchmark. To keep the number of samples the same for all algorithms, the RL training was divided into phases, as described in section V.C.

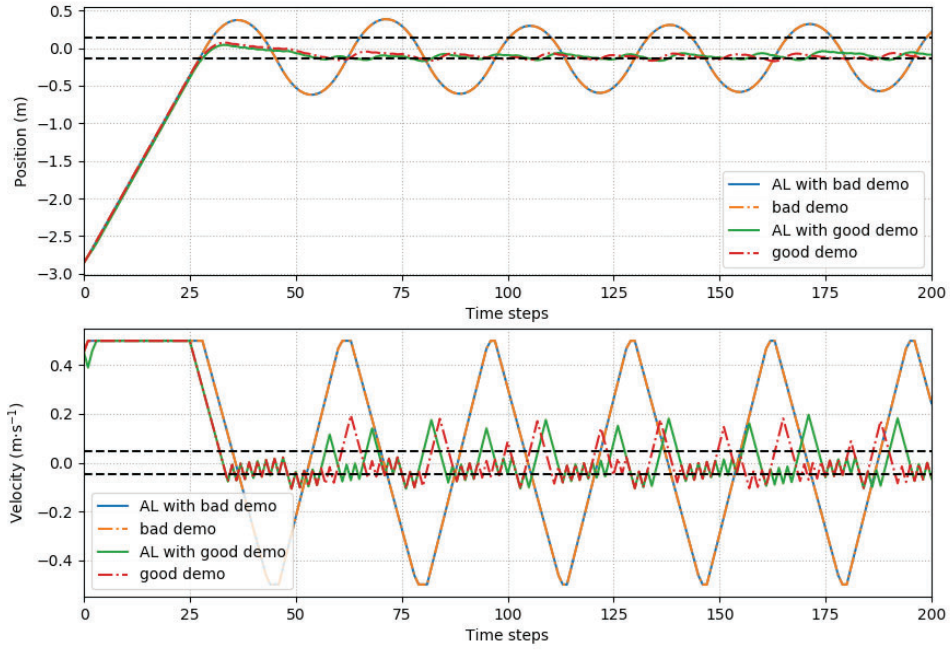


Fig. 3 Control history of quadrotor in one trajectory using the AL algorithm.

A. Time Traces

This section shows the changes of states over time steps to provide a general idea of the task and experiment setups. Figure 3 shows the time trace graph for one of the initial states when the AL algorithm is used. Four policies are compared in this graph: the AL policy using the bad demonstration and PFP (blue lines), policy corresponding to the bad demonstration (orange lines), the AL policy produced using the good demonstration and PFP (green lines) and the policy corresponding to the good demonstration (red lines). In each sub-figure, the margin between the two black dashed lines represents the area of the target discrete state. For position z, this area is $[-0.143, 0.143]$(m), and for velocity $\dot{z}$, this area is $[-0.045, 0.045](\mathrm{m} \cdot \mathrm{s}^{-1})$. As Figure 3 shows, the AL learns the policy that accomplishes the control task only when the demonstration is able to do the same. However, the AL-PFP combination always learns a very similar policy to the demonstrated policy.

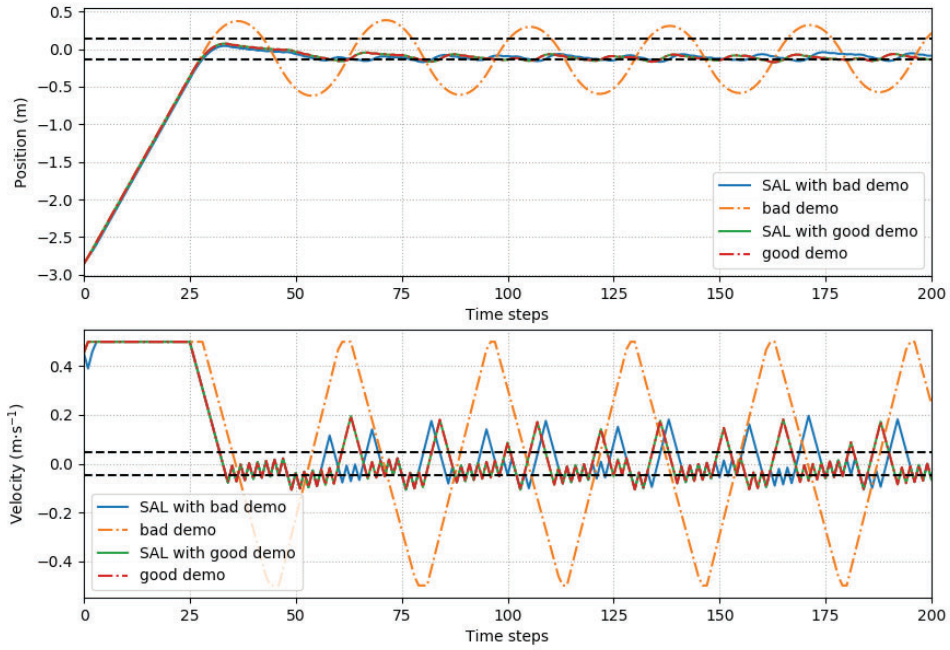


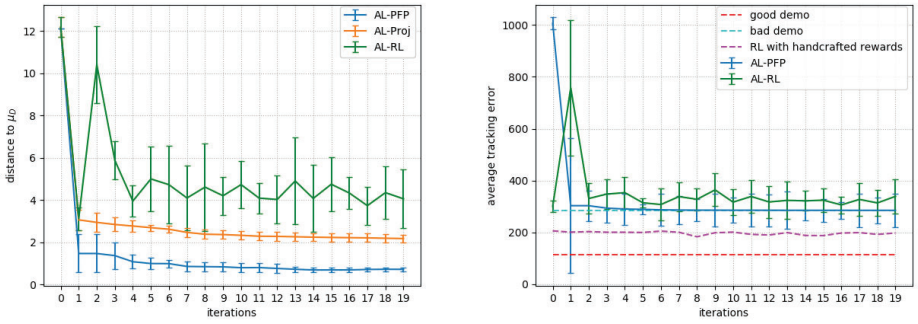
Fig. 4 Control history of quadrotor in one trajectory using the SAL algorithm.

Figure 4 shows the trajectory starting from one of the initial states, as produced by the SAL algorithm. The black dashed lines also define the area of the target discrete state. As this figure shows, SAL can learn a policy that can accomplish the control task based on either the good demonstration or the bad demonstration.

B. AL

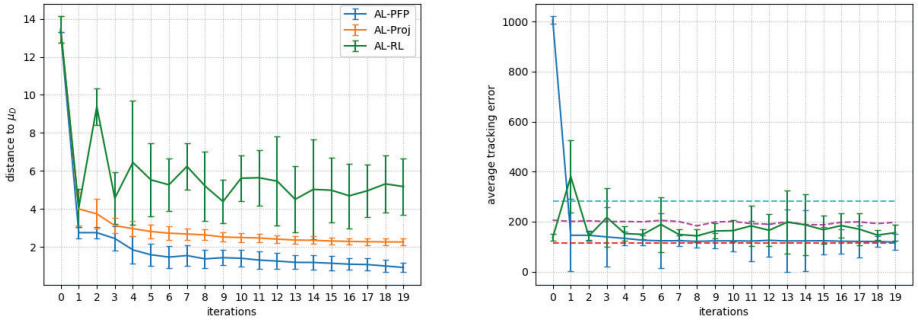
The AL using the original projection method and the AL with the PFP method are compared first, as shown in Figure 5. The orange lines (AL-Proj) show the policies produced by the original projection method with an additional post-processing step, as described in section III.A. The green lines (AL-RL) show the policies produced by the projection method without this post-processing step; these policies are thus the direct output of the RL inside the AL algorithm. The blue lines (AL-PFP) are the policies given by the AL with the PFP method. Figures 5a and 5c show the distances between policies' feature expectations and the feature expectation μ_E. Figures 5b and 5d illustrate the performance measurements of the policies taken directly from the RL and from the PFP method across 20 iterations, using the measurement defined by equation 9.

Figure 5a and Figure 5c show the feature expectations of policies given by AL-PFP is generally more similar to the demonstration's feature expectation. Figure 5b and Figure 5d show the AL-PFP combination also provides



(a) Feature expectation distances using the bad demonstration

(b) Performance measurements using the bad demonstration.



(c) Feature expectation distances using the good demonstration.

(d) Performance measurements using the good demonstration.

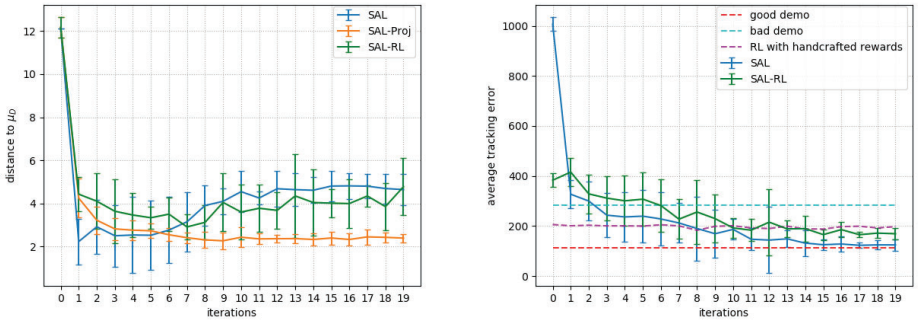
Fig. 5 Feature expectation distances and performance measurements of the AL as compared with an over-damped PD and an oscillated PD, averaged over 10 runs with a random initial policy. Note that the policy fusion projection gives feature expectations closer to those of the demonstrations, and the performance is better than RL policies in the AL.

policies that perform similarly to the demonstrated one. On the other hand, both the AL-RL policies' distance to the feature expectation of demonstration and their performance fluctuates throughout the 20-iteration training. The feature expectations of these policies are less similar to the feature expectation of demonstration; the performance of these policies is also not as good as the demonstrated policy. Although PFP moves the feature expectation closer to the μ_D and also gives policies that perform better, these two improvements are not related, as policies that are far from the demonstration in terms of feature expectations can still perform better than the demonstrated policy.

However, the AL-PFP combination is still unable to surpass the performance of the demonstration. The demonstration serves as the only input from the outside to directs the AL agent and does not change over iterations. The AL algorithm drives the RL to produce policies with similar feature expectations, but no better performance. Thus, AL with or without the PFP method cannot outperform the demonstration, as illustrated by Figures 5b and 5d.

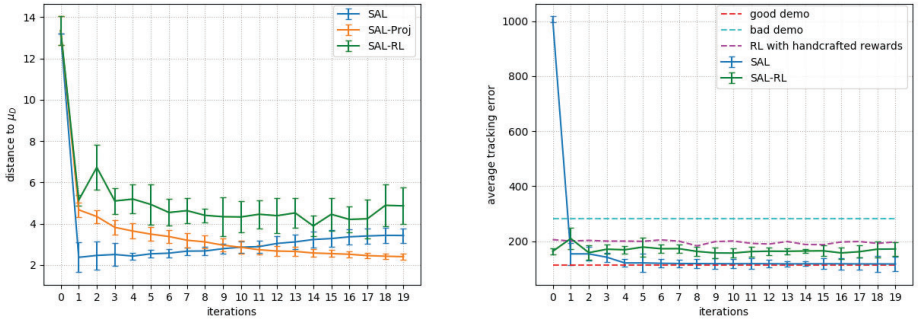
C. SAL

This section compares SAL using the original projection method (SAL-Proj) with SAL using PFP (SAL). Figure 6 uses the same performance measurements and the same illustration approach as the figure in the previous section. As shown in Figures 6a and 6d, the feature expectation of SAL policy strays from the demonstration feature expectation, being generally closer to the feature expectation of the SAL-RL policy with the exception of the first six iterations. The SAL-Proj maintains a small distance from the demonstration after the first few iterations. However, when trained with the same number of samples, SAL performs much better than RL in the case of the bad demonstration, performing almost as well as the good demonstration even when the input demonstration is the bad demonstration. If the good



(a) Feature expectation distances using the bad demonstration

(b) Performance measurements using the bad demonstration.



(c) Feature expectation distances using the good demonstration.

(d) Performance measurements using the good demonstration.

Fig. 6 Feature expectation distances and performance measurements of SAL, compared with a overdamped PID and a oscillated PID, averaged over 10 runs with random initial policy.

demonstration is used, similar performance is achieved with fewer iterations.

The SAL algorithm thus enables the agent to search for a better policy than the demonstrated one, finally breaking the upper bound of performance set by the demonstration in the original AL. In this experiment, it was notable that both the initial demonstration and the performance measurement played critical roles in the SAL algorithm development. The initial demonstration determines the starting point for SAL, while the performance measurement decides which trajectories should fill the demonstration buffer D_E.

Figure 6 also shows that the PFP method in SAL does not learn a similar policy to that of the demonstration, instead exploring and learning from trajectories with better performance. It should be noted that these newly found trajectories are extracted from the RL process, and thus these trajectories contains noise left by RL. However, the performance of the final SAL policy is better than that of the RL policy and the feature expectations of the SAL policies are drifting away from the demonstration's feature expectation. Therefore, by collecting and merging all past policies into a single more reliable option, SAL overcomes the effect of underperforming trajectories found by RL and learn the general policy behind these trajectories. Notice the blue line in Figure 6a changes differently to the blue line in Figure 6c. The distance of SAL policies using the bad demonstration dives down but then quickly increases and finally stays at the same level. The distance of SAL policies using the good demonstration also drastically decreases at the second iteration, but then it slowly climbs up. This is caused by the fact that the bad demonstration provides far less wellperforming trajectories than the good demonstration, and therefore the SAL algorithm using the bad demonstration has to rely heavily on trajectories found during the learning process. Those trajectories may contain noises, thus the process of overcoming these noises is illustrated by the increasing distance of feature expectations. Figure 6b and Figure 6d also support this observation: big improvements on the policies' performance are related to drastic change of feature expectations, and if the performance does not change, the feature expectation does not change too much. However, it is still not clear to what extent can SAL

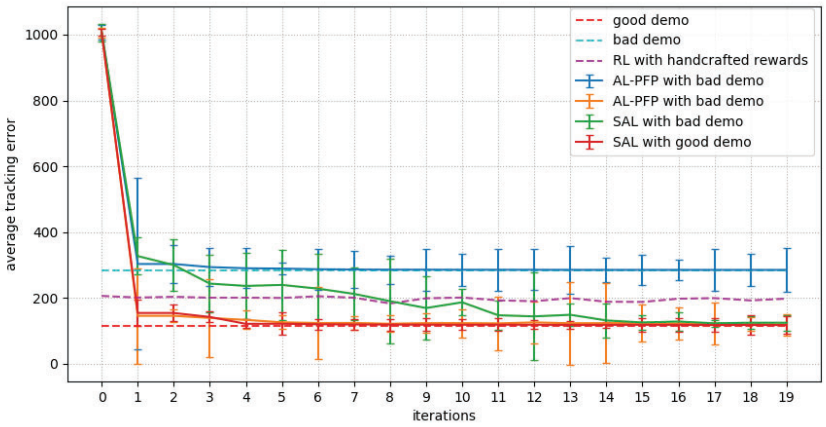


Fig. 7 The comparison of performance between the SAL and the AL

overcome the effect of underperforming trajectories.

A direct comparison of performance between SAL and AL can be found in Figure 7. When trained with the same number of samples, Q-learning with the handcrafted reward function designed in the previous section fails to perform better than the good demonstration but can outperform the bad demonstration. The AL with PFP also cannot perform better when provided with a bad demonstration. However, SAL reaches the same performance when using the good demonstration and the bad demonstration. SAL eventually performs as well as the good demonstration using the same amount of samples used by AL and RL, making SAL more sample-efficient than AL and Q-learning with this specific reward function. There are several other reward functions, however, so it is not clear whether or not other reward functions could improve the sample-efficiency of this RL algorithm.

VII. Conclusion

This paper shows that, for flight control, the policy learned by SAL outperforms the original demonstration. In terms of sample-efficiency, SAL surpasses AL and Q-learning with a handcrafted reward function. SAL has two major improvements: the PFP method merge the historic policies to produce one deterministic policy, and the trajectory replacement mechanism improve the demonstration with trajectories that have better performance measures.

The PFP method improves on the original projection method to increase the similarity of the learned policy. The PFP method is a majority vote mechanism for all historic policies trained in the AL algorithm, which asks every policy in the past to add probabilities to actions based on this policy's similarity to the demonstrator. This process generalises the information from the demonstrated policy throughout all past policies. Although the original projection method can also produce a distributed policy with higher similarity by means of a post-processing step, this distributed policy is not suitable for quadrotor control. Adopting the PFP method provides two advantages: the first is the fact that the produced policy is a real mapping from the state to the probability of action, the second is the produced policy is less affected by the trajectories that fail to fully express the underlying policy. The second advantage is crucial for the SAL algorithm.

The SAL algorithm introduces a trajectory replacing mechanism. That allows measurement of the performance of the trajectory by using external measurement. However, trajectories found by RL may contain noise, and thus those trajectories with noise cannot correctly indicate the policy of RL. But the effect of such trajectories can be reduced by the PFP method. Thus the PFP method enable SAL to generalise policies based on demonstrations. However it is not clear yet to what extent can PFP reduce the effect of those trajectories. The result of this paper thus suggest that the SAL algorithm can learn policies that outperforms the demonstrator while being less affected by the trajectories that cannot fully express their policies.

SAL is a model-free algorithm based on AL, being more sample-efficient than the original AL algorithm. SAL is also more sample-efficient than the Q-learning algorithm when a reward function similar to the one mentioned in this paper is used. In this paper, a quadrotor control task is used for experiments. It has been shown that SAL can be implemented to control a simplified quadrotor with high sample-efficiency. Experiments show the controller based on SAL outperforms

the demonstrations without the access to the quadrotor model. The potential of SAL is that the SAL algorithm can be implemented for flight control that involves complex dynamics and high dimensional state and action space. The SAL algorithm can greatly improve the efficiency of policy learning by learning from demonstrations obtained from other controllers or human pilots, and actively searching for better policy based on performance measurements. Thus using the resulting policies may improve the autonomy of flight control

In terms of future research, it would be interesting to examine the performance of different RL algorithms with different reward functions to compare their performance with that of SAL. The SAL method should also be tested in the situation when human provides demonstrations. Theories should be established to explain the relationship between any policy $\hat{\pi}$ produced by SAL and historic policies given by RL inside the SAL method. It should be possible to improve the state and action representation with function approximators to extend the state space and action space. The SAL algorithm in this paper uses discrete action space, future research should investigate how to use continuous action space to improve the performance of SAL-based controller. With the aforementioned research being conducted, the controller based on SAL can be implemented for more complex flight control problems and eventually the flight control problems in real life.

References

[1] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, MIT press, 2018.

[2] Ng, A. Y., Russell, S. J., et al., "Algorithms for inverse reinforcement learning." *Icml*, Vol. 1, 2000, p. 2.

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

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

[5] Boularias, A., Kober, J., and Peters, J., "Relative entropy inverse reinforcement learning," *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 182–189.

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

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

[8] Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S., "Path integral guided policy search," *2017 IEEE international conference on robotics and automation (ICRA)*, IEEE, 2017, pp. 3381–3388.

[9] Syed, U., and Schapire, R. E., "A game-theoretic approach to apprenticeship learning," *Advances in neural information processing systems*, 2008, pp. 1449–1456.

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

[11] Abbeel, P., *Apprenticeship learning and reinforcement learning with application to robotic control*, Stanford University, 2008.

[12] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al., "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, No. 7540, 2015, p. 529.

[13] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al., "Deep q-learning from demonstrations," *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] Gibiansky, A., "Quadcopter dynamics, simulation, and control," *Andrew. gibiansky. com*, 2012.

[15] Staples, G., "Propeller Static & Dynamic Thrust Calculation," , 2015. URL `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm`.

II

Literature Review & Preliminary Analysis

2

Literature Review

This chapter will first concentrate on some of the most important RL methods that have the potential to work with Learning from Demonstrations (LfD) methods, including value-based methods and policy gradient methods. Then LfD methods that learn reward functions will be introduced. The strength and weakness of these methods will also be discussed in the context of flight control. Chapter 2 and Chapter 3 are based on the work that was previously graded for AE4020 (Literature Study).

2.1. Reinforcement Learning Preliminaries

Reinforcement Learning simplifies real-world problems into a Markov Decision Process (MDP), which is a mathematical description of a series of sequential state-action pairs and rewards. In general, the MDPs describe a process in which agents can choose their actions a_t at each time step at state s_t. At the next time step, the agent will move to the next state s_{t+1} with the probability given by *state transition matrix* $p(s_{t+1} \mid s_t, a_t)$. In the meantime, the agent could receive a reward based on the state it is going to. Thus the transition probability with reward is defined as $p(s_{t+1}\, r_t \mid s_t, a_t)$.

Reinforcement Learning is a learning process that tries to find the optimal policy π based on the agent-environment interactions as shown in Figure 2.1. During the learning phase, the agent as a part of the MDP makes moves and receives rewards. After that these rewards are compared and finally a policy which can maximize the rewards is computed. The advantage of RL is that it does not require full knowledge of the environment instead the agent receives and collects information through the interactions with the environment [1]. In flight control, the dynamics of plant is often difficult to be accurately modeled, hence more research has been put on model-free control such as RL.

The problem of RL is it requires amount of samples of agent-environment interactions before producing a good policy. With the development and implementation of intelligent control, the complexity of control tasks has been risen up to a higher level. The number of samples required for these tasks has also increased. Improving the sample-efficiency has

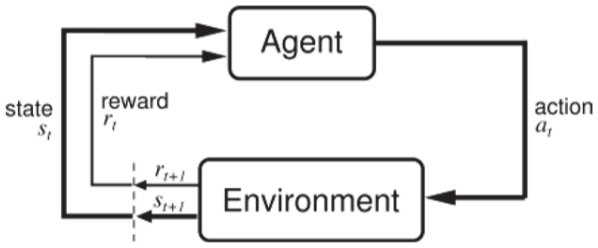


Figure 2.1: The agent-environment interactions in reinforcement learning[1]

become an urgent task.

2.2. Temporal-Difference Learning and Q-Learning

An important category of Reinforcement Learning is Temporal-Difference (TD) [5] Learning. The TD method combines the strength of the Dynamic Programming (DP) [6] and Monte Carlo method [7]. In DP, it is assumed that the dynamics of the environment is already known and can be represented by a transition model. In the Monte Carlo method, a model is not needed but the estimations can not be updated as in the DP until the agent reached the terminal state. The TD method updates estimations without waiting for the final outcomes. It also learns from its experience with the environment without any access to environment models. In this section we will deploy the basics of the TD method and then introduce one of its variant: Q-Learning. This is because our Learning from Demonstration method is based on it.

The TD method is a combination of DP and the Monte Carlo method, but it inevitably introduce bootstrapping into its algorithm. The estimation of the current state value is updated by the state value at the next time step. One simplest form of the TD update is

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]. \tag{2.1}$$

The update rule shows that the target which this algorithm is trying to get is a current state value estimated by the next state value and the immediate reward. This is called TD(0) as it is a special case of TD(λ). The algorithm for TD(0) value estimation is shown in Algorithm 1.

This algorithm is called TD(0) because its update target is the estimated value of the current state S_t given immediate reward R_{t+1} upon the transition to the next state S_{t+1}. This formation can be extended to TD(λ) where the update target becomes a state value estimated by λ steps of accumulated discounted rewards upon transition to state $S_{t+\lambda+1}$. By taking into account more and more future rewards, the bootstrapping at each time step will be more and more accurate until the λ reaches time horizon and then TD(λ) is equivalent to the Monte Carlo method.

The TD method has many variants, one based on TD(0) is called Q-Learning [8]. Q-learning updates the action-value function, thus it is suitable for solving control problems. Furthermore, compared to other TD methods, Q-learning greatly increases the convergence speed and facilitates the analysis by the directly estimation of Q-values without following a policy.

Algorithm 1 Tabular TD(0) value estimation [1]

Input: policy π to be evaluated, step size α
Initialize $V(s)$ for all states
for each episode **do**
  initialize S
  while S is *not* terminal **do**
    generate action A from given policy π
    take action A, observe reward R and S'
    update $V(S_t)$ by $V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$
    $S \leftarrow S'$
  end while
end for

The minimal requirement of correct convergence is any state action pair continues to be updated. The update rule of this algorithm is defined as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \right] - Q(S_t, A_t). \tag{2.2}$$

Similar to TD case, Q-Learning also has a procedural form which is shown in Algorithm 2.

Algorithm 2 Q-Learning (off-policy TD control) [1]

Input: step size parameter α
Initialize $Q(s, a)$ for all state action pairs
for each episode **do**
  initialize S
  while S is *not* terminal **do**
    generate action A using policy derived from Q (e.g., ϵ-greedy)
    take action A, observe reward R and S'
    update $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \right] - Q(S_t, A_t)$
    $S \leftarrow S'$
  end while
end for

Another variant of TD(0) is SARSA. It is an on-policy TD control algorithm. Different from Q-Learning, the estimation of the action-value function in SARSA follows the policy which has been used. Although Q-Learning learns the value of the optimal policy, its online performance is worse than that of SARSA which learns the roundabout policy [1].

2.2.1. Deep Q Learning

To deal with the high dimensional situation for Q-learning, function approximators are usually used to estimate action values. One popular choice is the Artificial Neural Networks (ANNs). An ANN with one hidden layer and non-linear activation function can theoretically approximate any non-linear function to any degree of accuracy [9]. Deep Q Network (DQN)[10] uses ANNs to extract features from high dimensional state space and estimate

the Q values. DQN was evaluated in several Atari games and the performance shows it even outperforms human in three games.

Updating neural networks solely based on current experience results in unstable policy [11]. The solution to this is the experience replay buffer where the agent can randomly extract old experience from. The target network is also separated from the action value networks that is constantly updated. In the two networks that the DQN has, action value network generates the action and update itself by gradient descent with sampled batch of transitions from the experience replay buffer, while the target. The parameters of the target network is only regularly updated by copying the other network's parameters.

The original DQN has several disadvantages. First of all, it overestimates Q-values because the action is chosen by the action value network which is unstable for it is been constantly updated, thus errors at each update are propagated through the entire learning process. The Double Q-Learning [12] mitigates the overestimate problem by choosing actions based on the target network.. Another disadvantage is the experiences stored in the replay buffer can not been efficiently selected. Prioritized Experience Replay [13] solve this problem by ranking the old experience with the corresponding TD-errors. The larger the TD-error will be, the higher the rank of the experience is. The transitions in the replay buffer are then extracted more efficiently by selecting those with higher ranks.

2.3. Policy Gradient Method

The methods discussed in previous sections are all state value or action value methods as they measuring usefulness of states and actions with actual values. Another possible choice is parameterizing policy so that actions can be selected without consulting a value function although a value function may still be used to estimate policy parameter. The policy parameterized by vector $\boldsymbol{\theta}$ can be written as

$$\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a \,|\, S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}. \tag{2.3}$$

The action selection for paramterized policy setting becomes choosing the action with highest numerical preference $h(s,a,\boldsymbol{\theta})$. One example of using preference is using it with exponential soft-max:

$$\pi(a|s,\boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}}. \tag{2.4}$$

Numerical preference function $h(s,a,\boldsymbol{\theta})$ can be computed in various way, both linear methods or non-linear methods, such as ANNs as in AlphaGo system[14], can describe this function. To simplify the discussion, preference functions are all linear in features, in other words, they can be formed as

$$h(s,a,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}(s,a). \tag{2.5}$$

There are two advantages of parameterizing policy according to soft-max distribution [1]. The first advantage is that parameterized policy can approach to deterministic policy, whereas with ϵ-greedy policy there are always ϵ chance that policy would not choose optimal action. A second advantage is this setting enables the arbitrary selection of actions as in some cases stochastic policy may be the best.

To ensure policy improvement upon changing the policy parameter, Sutton et al. developed the policy gradient theorem [15]. This theorem establishes that for episodic tasks with performance measurement $J(\boldsymbol{\theta}) \doteq v_{\pi_\theta}(s_0)$, the gradient of $J(\boldsymbol{\theta})$ can be written as

$$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}). \tag{2.6}$$

The theorem describes that, for episodic case, the gradient of performance measurement which is represented by the value of the first state, can be measured by the gradient of parameterized policy with action value function and a distribution of state μ. Combining policy gradient theorem with Monte Carlo control gives the REINFORCE algorithm [16] for episodic control as shown in Algorithm 3.

Algorithm 3 REINFORCE: Monte Carlo Policy-Gradient Control [1]

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size α
Initialize policy parameter $\boldsymbol{\theta}$
for each episode **do**
  generate an episode S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_t, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  for each step of the episode $t = 0, 1, ..., T-1$ **do**
    update $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})'$
  end for
end for

It is also possible to extend REINFORCE with a state value baseline, $\hat{v}(S_t,\mathbf{w})$. REINFORCE with baseline improve the convergence speed but, similar to the original REINFORCE, it still waits for the episode to be finished in order to update parameters. Actor-critic methods improve the learning speed and usefulness for both online and continuing problems by combining policy gradient and one-step bootstrapping temporal-difference methods. The resulting algorithm is called One-step Actor-critic [17] for control which is a fully online, incremental algorithm updates both policy parameter $\boldsymbol{\theta}$ and value parameter $\mathbf{w}$ at each time step.

In recent years, the policy-gradient has been implemented in many successful algorithms. Trust Region Policy Optimization (TRPO) [18] is an algorithm that uses policy gradient. It is shown that this algorithm prevents unstable variations in policy networks during training by always keeping the new policy parameters in a reasonable distance to the old ones [18]. The algorithm also propose a exploration strategy that first derive a set of trajectories and then create branch trajectories from them. However TRPO uses Kullback-Leibler divergence as a condition when optimizing its objective function, and thus makes it difficult to code. The Proximal Policy Optimization (PPO) [19] is relatively easy to code and yet preserves most features in TRPO. Although simpler in the form, PPO outperforms TRPO in many benchmark problems as shown in figure 2.2. Deterministic Policy Gradient (DPG) [20] methods uses actor-critic framework and deterministic policy gradient theorem [20] to extend the RL algorithms to continuous state and action space and enhance the stability. Deep Deterministic Policy Gradient (DDPG) [21] evaluates actor networks and critic

networks to approximate stable policy parameters. These are the state-of-the-art RL algorithms that can be used as benchmarks for future research.

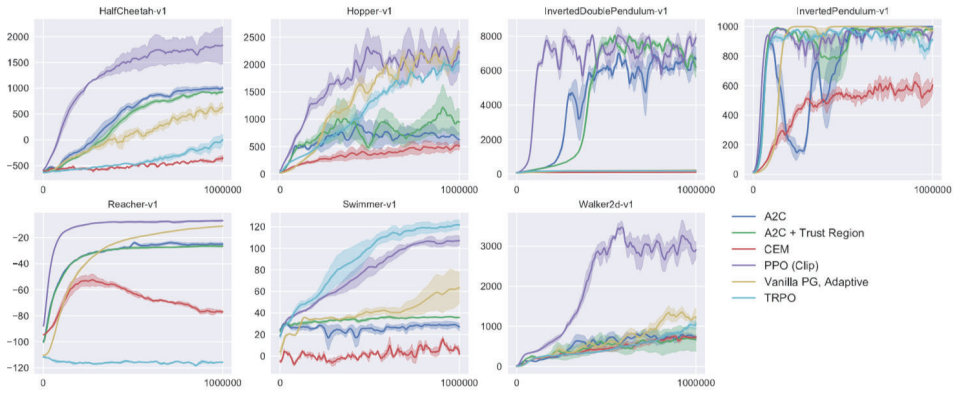


Figure 2.2: Comparison of several algorithms on several MuJoCo environments. PPO outperforms TRPO in every simulation [19].

2.3.1. Actor-critic and Deep Deterministic Policy Gradient

It has been shown that although the Policy Gradient methods have the potential to produce continuous actions, in natural Policy Gradient methods the step size of an update is actually controlled by the action value and this causes unstable behaviors for the agent. Actor-critic [22] proposes an approach that the step size of a policy update is controlled by the TD-error of the action values. For example, the update rule for the Actor parameters $\boldsymbol{\theta}$ of the one-step Actor-critic can be written as

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha\delta\nabla\pi(a|s,\boldsymbol{\theta}) \tag{2.7}$$

Where δ is the TD-error of the Critic and α is the learning rate. Actor-critic has two RL learners learning from experience in parallel, one is the Actor based on the Policy Gradient, the other is the Critic based on a value-based RL. The Critic can steer the direction of the Actor by providing information regarding the goodness of certain actions.

The Deep Deterministic Policy Gradient (DDPG) [21] is a combination of the Actor-critic and DQN. It efficiently learns from high dimensional states and produces deterministic action from continuous action space. The structure of DDPG is similar to DQN: the Actor and the Critic both consists of a target network and a main network that is been constantly updated. DDPG was evaluated in several OpenAL Gym environment and was compared against the original Deterministic Policy Gradient method, the conclusion was target networks is crucial for DDPG as it greatly increase reward received and the stability [21].

2.4. Reinforcement Learning for flight Control

In this section some implementation of RL algorithms in aircraft control will be introduced, motivating the answer to the first research question: Which state-of-the-art model-free Reinforcement Learning methods have the potential to be implemented in flight control? For flight control, many RL algorithms have been implemented, ranging from the earlier actor-critic controller and the newly developed deep RL controller. In general, RL controller can

be categorized into the model-based method and the model-free method. The difference of model-based and model-free RL is the agent's awareness of model. In model-free RL, agent interacts with the environment to obtain feedbacks which in turn form the agent's policy. But in model-free RL the agent either takes the knowledge of model as a prior or obtains the model through system identification techniques while exploring the environment. RL algorithms have also been implemented in various aircraft models, such as fix-wings [23] and quadrotors [24, 25]. Different aircraft may needs different RL algorithms. But for most of the aircraft, the model is difficult to obtain, therefore model-free RL is more often used.

The model-based method requires knowledge of aircraft's dynamics model a prior. Usually the accuracy of model is directly related to the performance of control. For example, in Ferrari's paper [23], a learning scheme based on the dual heuristic adaptive critic separates the learning process into the the offline phase and then the online phase. The offline phase trains the actor and critic at given operating points by the given gain matrices. In this phase, a prior knowledge builds the basis of actor and critic networks. This knowledge of operating points and dynamics covers a large part of all potential operating points and the overall dynamics but is also limited. This step is to only determine the size and weights of the neural networks in the actor and critic. The second phase expands the knowledge of operating points and dynamics in an online manner. Actor and critic are updated incrementally during the second phase to improve the control response based on the actual state of the plant accounting for the estimated state and action from the first phase. As the online training goes on, this expansion grows bigger, and more and more knowledge is incorporated into the controller. Finally the controller is able cover both the dynamics and operating points that are already included in the a prior knowledge and experience for the first time. In genarl, Ferrari's method incorporates the knowledge of a existing controller while learning from the dynamics experienced online to reduce gradually the effect of disturbance and unmodeled dynamics. This method was implemented in a six-degree-of-freedom simulation of business aircraft, and the result showed that, in a tracking task, the performance of controller was improved even some unmodeled dynamics and unexpected disturbance are experienced for the first time [23]. Figure 2.3 shows that, in the presence of parameter variations of the plant's dynamic model, Ferrari's adaptive actor-critic control performs better than the fixed-parameter neural networks control but cannot outperform the neural networks control with perfect modelling. Ferrari's method proposes a novel approach to the RL control. Although the dynamics model and other knowledge of the plant can be perfected during the online learning phase, it is still essential that some a prior information must be provided in the offline learning phase, which indicates that amount of works around system identifications must be carried out before the offline learning phase. In fact, most model-free RL control method require the identification of aircraft dynamics model before learning.

Another model-based RL implementation for flight control is shown in [24]. This implementation demonstrates the difference of a integral sliding mode controller and a model-based RL controller in quadrotor control. The RL controller is built based on the LWLR method [26] taking into account the estimation of Gaussian noise. The model of the quadrotor assumed the disturbance can be estimated by Gaussian noise. Although the true disturbance may not be perfectly Gaussian, but they found this assumption was adequate for

this implementation [24]. The results of this implementation were shown by the performance of the integral sliding mode controller and the RL controller in a tracking control task. Although they both responded similarly to step inputs but the RL controller was advantageous in estimating the Gaussian noise [24].

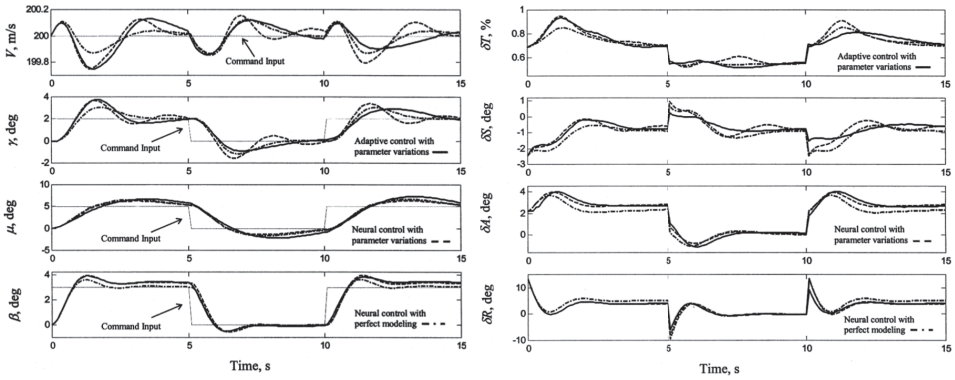


Figure 2.3: Comparison between the adaptive and the xed-parameter neural controllers in the presence of parameter variations [23]

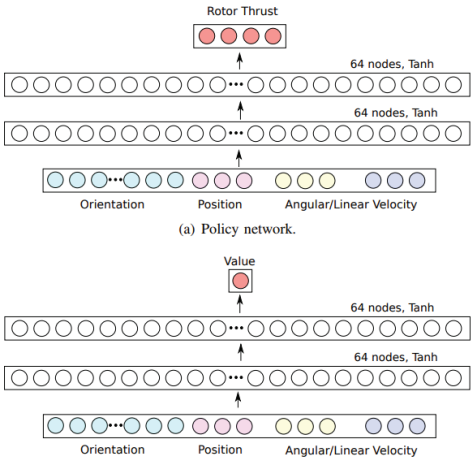


Figure 2.4: The two neural networks in DDPG for quadrotor control [25]

The model-free approach for aircraft control is a fast-developing branch in RL. Thanks to the advanced techniques for Neural Networks, efficient algorithms has been developed to solve the highly complex, nonlinear aircraft control problems. For example, the ideas of TRPO and DDPG were adapted by Hwangba *et al.* [25] for quadrotor controller.The training algorithm is based on DDPG and the same exploration strategy in TRPO. The structure of two neural networks in this implementation is shown in figure 2.4, the first networks, the *actor*, generates four thrust signals, while the second networks, the *critic* finds the update of the first networks' parameters. These two Neural Networks consists of DDPG in this implementation. The exploration strategy is adapted from the exploration strategy in TRPO, which is exploring different trajectories sections by sections, and inside one section the policy remains the same until the next section. This implementation simplified this exploration strategy into expanding the initial trajectory with branch trajectories extended from junction points. This simplification greatly helped the exploration of RL algorithm

and reduced the learning time. Computational resources were also organised and optimized to increase the computational efficiency. This implementation was validated both in simulation and physical experiment. The task was about training TRPO, DDPG and the proposed method to track waypoints. The figure 2.5 shows the simulation result that TRPO and DDPG was not able to converge to adequate performance in reasonable amount of time and he proposed algorithm outperformed the other two. This quadrotor implementation of RL algorithms shows the novel RL techniques such as TRPO and DDPG can learn to stabilize and control a aircraft with complex dynamics even under extreme conditions such as a upside-down position [25]. In this implementation, the architecture of Neural Networks was very simple: three layers Neural Networks with one hidden layer to map the actions and states. Training techniques mentioned before also greatly helped the learning efficiency. But due the complex nonlinear dynamics of quadrotor and large state-action space, the training in simulation and real flight still faced high difficulty.

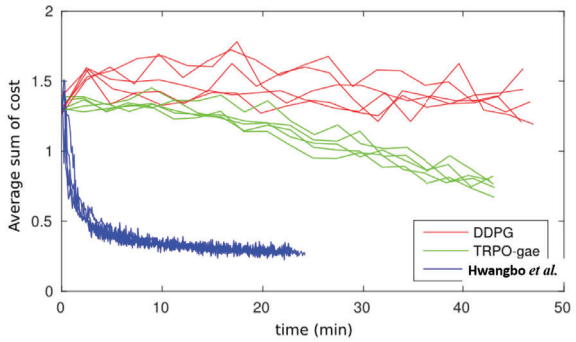


Figure 2.5: Learning curves for optimizing the policy for three different algorithms [25]

This section have presented some implementations of RL algorithm in aircraft control to answer the first research question. The most distinctive categories are model-based and model-free RL, and algorithms from both of these categories have been implemented in aircraft control. Many of these implementations are about simulations, and the most representative is the simulation of quadrotor. A quadrotor has highly nonlinear dynamics. Simulating such complex dynamics demands a large amount of computational resources. Therefore simplifications may be required. Literature mentioned the complexity of simulation can be reduced if some parts of the aircraft dynamics can be neglected. A common way to neglect some dynamics is focusing on the dynamics in only one dimension while ignoring others, such as the simulations in [23, 24]. In addition, many implementations such as [23, 24, 25] choose position or speed tracking task as the control task. Therefore these tracking task will also be selected as the task for LfD algorithms.

2.5. Learning from Demonstrations Preliminaries

The idea of LfD is to train an agent that can map action to its state based on knowledge learned from expert's trajectories. Then the agent can make a decision to encounter other situations. LfD can be seen as a subset of supervised learning in the sense that, same as supervised learning, training data D(i.e., expert's trajectories) with correct labels are used to train an approximation to the agent that produce similar data (agent's trajectories). In LfD these training sets usually contain the states that have been observed and actions associated with each of the states. LfD problems usually can be formed as follows. The environ-

ment consists of states S and actions A and transition probability $P(s' \mid s, a)$ defining the possibility of going to state s' from state s upon taking action a. For simplicity we assume state S is observable. The LfD algorithms try to learn from the expert's demonstration and formalize the obtained knowledge into policy. Then the policy can be executed in the real environment [2].

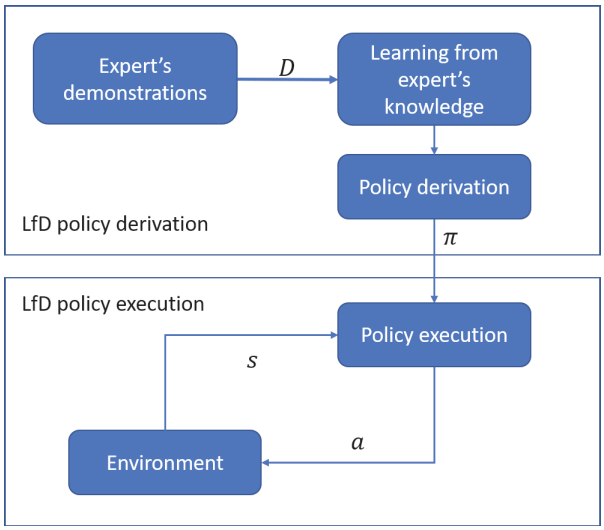


Figure 2.6: LfD learns policy from demonstrations

LfD has many names: Learning by Demonstration (LbD), Programming by Demonstration (PbD), Learning by Experienced Demonstration, Assembly Plan from Observation, Learning by Showing, Learning by Watching, behavioral cloning, imitation, mimicry, and the one we use, Learning from Demonstration (LfD) [2]. For each name, the scenario could be slightly different although they use the same idea. For example, Programming by Demonstration is a technique which is usually applied in teaching robots skills to finish a task like picking up a box. Behavioral cloning often refers to the learning techniques with which agent can reproduce movements and trajectories from teacher demonstrations. For each name of these techniques the learning agent might want to achieve their goals in different levels, the lower level task can reproduce a single signal and the higher level task can be finishing a complex mission that consists of a series of control signals.

LfD can be subdivided into Mapping, model learning, and planning, as shown in Figure 2.7. Mapping is a method that directly approximates the model that link actions to states. Model learning on the other hand, first extracts information of the environment including transition functions and reward function and then derive policy based on that information. Planning is the method that uses demonstrations and possibly user's intentions to train rules that relate pre- and post-conditions (i.e. the states before and after actions) to each actions to produce policy [2].

2.6. Learning from Demonstrations: Mapping

Mapping is essentially a technique that aims to reproduce demonstrator's underlying policy, and to generalize over the available training data so that solutions to states that similar

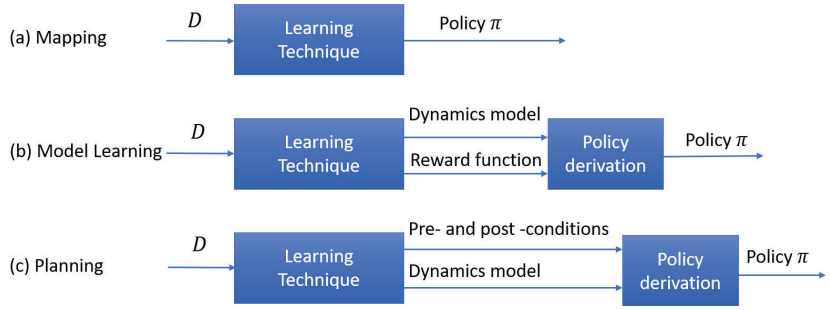


Figure 2.7: Three subdivisions of Learning from Demonstrations, adapted from [2]: (a) mapping learns policy by linking states to actions based on demonstration data (b) model learning learns dynamics of the world and reward function (c) planning learns the relation between pre-, post-conditions and actions, and possibly world's dynamics

to states in training data but not encountered before may be found.

2.6.1. Mapping Methods without Reinforcement Learning

Some mapping methods do not work with RL algorithms but still produce good policy. These mapping methods can be used to generate both low-level motion control and high-level behaviors. Chernova et al. [27], simulated a low-level control for car driving using a confidence-based policy learning from the demonstrations method with a Gaussian Mixture Model (GMM). In their algorithm, the policy was parameterized by means and variances of a GMM, and then the Expectation-maximization algorithm was used to updates these parameters. The preferences to actions at different states are modeled by the GMM. This algorithm first trained the GMM with non-interactive demonstrations and then enabled the agent actively ask the human demonstrator for instructions when certain confidence requirement is not met. Although their result showed that in a car driving simulation their algorithm reduces learning time, the agent still performs bad at the early stage of training, and the algorithm requires additional human-in-the-loop training in order to train a desirable policy.

For high-level behaviors training, Rybski et al. [28] trained a Pioneer robot for a box sorting task using a Hidden Markov Model (HMM) that classifies demonstrations into gestures. The gestures captured by cameras were translated by a HMM to choose a label of action among several pre-defined actions. The result showed that with only a few pre-defined actions, the robot could reliably make the correct choice. But the number of actions are very limited and the agent is design for only several tasks like grasping, releasing and following.

Mapping can also be used for flight control. An experiment [29] with a fight simulator using decision trees used human business jet pilot data to train an autopilot. The log files of real flights were used to construct the training data. The agent was trained to finish a series of actions for takeoff. The agent could finish the takeoff task but it could hardly perform any other tasks.

2.6.2. Mapping Methods with Reinforcement Learning

Deep Q-learning from Demonstrations (DQfD) [30] is a combination of the famous Deep Q-network (DQN) [11] with LfD techniques. DQN is an Artifical Neural Network with con-

volutional layers. The structure of DQN is shown in Figure 2.8, it has two hidden layers, one input layer and one output layer. As a example, DQN can be trained to play Artari games and its performance surpass many reinforcement learning algorithms for playing the same games. This algorithms can be easily tuned for other purpose as well. DQN extract features and approximate the optimal Q-values[8] in large state space using deep neural networks [31].

DQfD adds additional step in DQN which initialize the algorithm with a demonstration data set. The demonstration data set is consist of input states and corresponding output actions. Then the parameters of the network are trained with these state-action pairs , this procedure is shown in the box below. In the algorithm, θ is the parameter vector of target DQN for Q-value approximation, D^{replay} is a data set to store both demonstration data and training trajectories in order to maintain stability. DQfD has been trained to play three Artari games, the result, as illustrated in figure 2.9, shows DQfD managed to outperforms Prioritized Dueling Double Deep Q-Networks (PDD DQN) and an imitating agent that learns its policy without interacting with the environment.

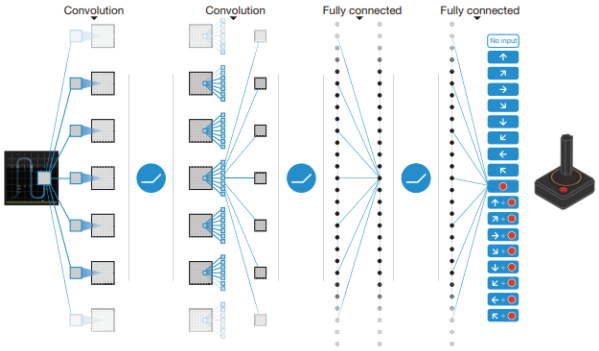


Figure 2.8: DQN: deep neural network with two hidden lays. The input is a stack of 4 consecutive frames of video games, the output is actions of joystick[11]

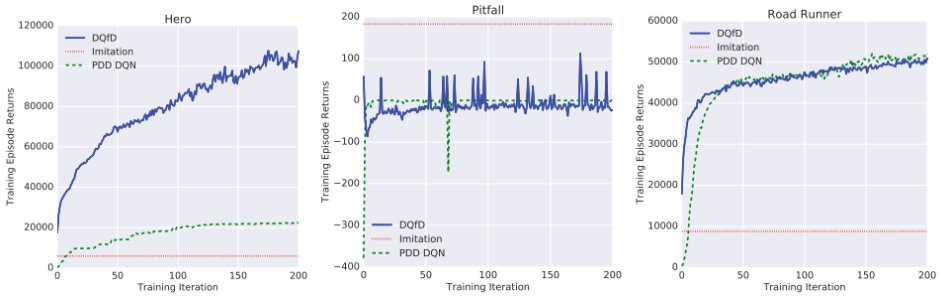


Figure 2.9: On-line scores of the algorithms on the games of Hero, Pitfall, and Road Runner, trained with DQfD, PDD DQN, and an imitating agent[30].

One-shot Imitation Learning [32] extends the capability of deep RL algorithms, it learns policy fast for multiple tasks with only one learning-from-demonstrations procedure. In One-shot Imitation Learning, the network architecture is consist of three parts, as shown in figure 2.10, including a demonstration network, a context network, and a manipulation

Algorithm 4 Deep Q-learning from Demonstrations [30]

Input: D^{replay}:initialized with demonstration data set, θ: weights for inital behavior network (random), θ': weights for target network (random), τ: frequency at which to update target net, k: number of pre-training gradient updates
for steps $t \in \{1, 2, ..., k\}$ **do**
  Sample a mini-batch of n transitions from D^{replay} with prioritization
  Calculate loss $J(Q)$ using target network
  Perform a gradient descent step to update θ
  if $t \bmod \tau = 0$ **then**
    $\theta' \leftarrow \theta$
  end if
end for
for steps $t \in \{1, 2, ...\}$ **do**
  Sample action from behavior policy $a \sim \pi^{\epsilon Q_\theta}$
  Play action a and observe (s', r).
  Store (s, a, r, s') into D^{replay}, overwriting oldest self-generated transition if over capacity
  Sample a mini-batch of n transitions from D^{replay} with prioritization
  Calculate loss $J(Q)$ using target network
  Perform a gradient descent step to update θ
  if $t \bmod \tau = 0$ **then**
    $\theta' \leftarrow \theta$
  end if
  $s \leftarrow s'$
end for

network. The demonstration network learns from demonstration data of multiple tasks and produces signals that enable the context network to discerns different situations. The context network compares the current state with signals from the demonstration network and determines which scenario is matching to the current state, this procedure generates a context signal passing to the manipulation network. Finally, the manipulation network decides the action based on the context it has been given. The One-shot Imitation Learning essentially extends the deep RL algorithms from being capable of generating low-level control to high-level behaviors. Although current experiments are about simple block-stacking tasks and do not use high-dimension input such as images, this framework does have the potential of being able to perform more end-to-end tasks.

Mapping methods have shown their strength in many domains. In general, mapping approximates the function of states to reproduce demonstrator's actions. Although DQN has been successfully implemented for UAVs [33], its variate DQfD has not been used for the same purpose. It should be possible that DQfD can also be implemented for flight control.

2.7. Learning from Demonstrations: Planning

Planning methods form the policy as a sequence of state-action pairs which lead the agent from the initial state to the goal state [2]. In other words, it link the state sequence to action

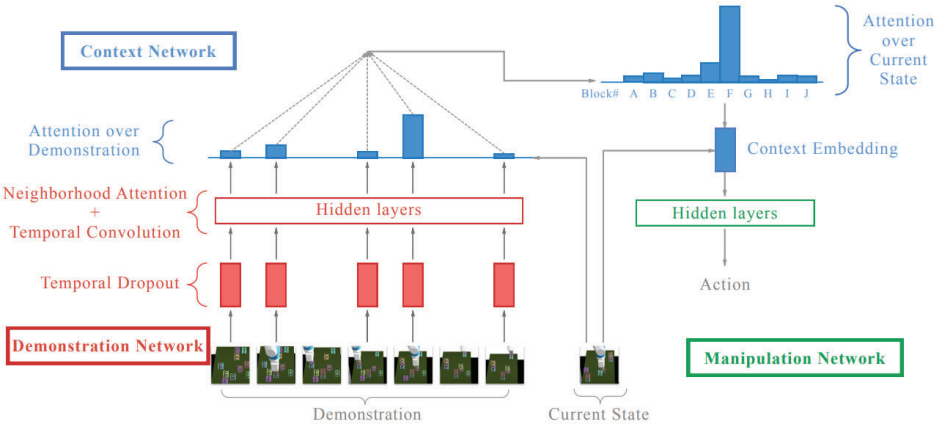


Figure 2.10: The architecture of One-shot Imitation Learning [32].

sequence to make a plan for the agent. In planning methods, actions are usually defined by pre- and post-conditions of states, certain states must be established before and after the action be performed.

One example of using planning methods is training a humanoid robot to finish a sequential repetitive ball-picking task [34]. The algorithm recognize the pre-conditional state in order to find the optimal action, and it is able to extract plans based on only two demonstrations. However, the algorithm is task specific and thus has limited potential of being transformed to other domains. Planning can also be used for model learning, Garland et al. present a method using sequential high-level human instructions and annotations to learn a task specific hierarchical model [35].

In general, planning methods are limited to specific tasks due to the information extracted from demonstration is limited to plans with conditions. It is often hard to generalize plans over a large state space because planning methods usually require amount of sequential state-action combinations series for the training over a small subset of states. The use of planning methods is also limited to robot understanding of instructions and model structures revealing and hardly be extended to autonomous multi-tasks domain. This limitation exclude the choice of planning methods as flight control usually expects the better autonomy.

2.8. Learning from Demonstrations: Model-Learning

Model-Learning approach to LfD learns a model of the world, and derives a policy π from the model. The model usually consists of transition model and reward function, but for RL algorithms introduce in the previous sections, reward functions is crucial as the transition model of the environment can be explored and understood by RL.

A good reward function should be able to highlights the target state while illustrating the most suitable paths. Defining the reward function is a non-trivial task but sometimes good reward values are not obvious. Inverse Reinforcement Learning (IRL) is a category of algorithms that learn reward function.

2.8.1. Inverse Reinforcement Learning

IRL constructs the reward function by learning from information of the transition model or demonstrations. IRL problem is to find a reward function that forms the best explanation of the observed behaviors. IRL problem under the RL framework can be formed as follows [36]:

- **Given** 1) measurements of an agent's behavior over time, in a variety of circumstances, 2) if needed, measurements of the sensory inputs to that agent; 3) if available, a model of the environment.
- **Determine** the reward function being optimized.

The MDP is a tuple $\left(S, A, \{P_{sa}\}, \gamma, R\right)$ where

- S is a finite set of N states,
- $A = \{a_1, ..., a_k\}$ is a set of k actions,
- P_{sa} are state transition probabilities upon taking action a in state s,
- $\gamma \in [0, 1)$ is the discount factor,
- $R\ S \rightarrow \mathbb{R}$ is the reward function bounded by a maximum value $R_{\max}$.

IRL problems can be divided into the IRL in finite state spaces and the IRL in large state spaces. The first finite state space setting can be solved with the explicit knowledge of the transition probabilities P_{sa}. The first step is to express reward function with transition probabilities.

Suppose the reward function can be written in the form of a vector $\mathbf{R}$, the transition probability matrices upon taking the optimal action a_1 in every state and the sub-optimal action a in every state are $\mathbf{P}_{a_1}$ and $\mathbf{P}_a$, the reward R satisfies [37]

$$\left(\mathbf{P}_{a_1} - \mathbf{P}_a\right)\left(\mathbf{I} - \gamma \mathbf{P}_{a_1}\right)^{-1} \mathbb{R} \succeq 0. \tag{2.8}$$

The reward function R can be found by solving this equation. However this is a ill-posed problem since there are many choices of R meets the criteria: 0 and arbitrary values close to 0 are always solutions. This can be solved by adding a penalty term. Finally, the search for R can be formed as a linear programming problem, the optimization objective is [37]

$$\text{maximize} \sum_{i=1}^{N} \min_{a \in \{a_2, ... a_k\}} \left\{\left(\mathbf{P}_{a_1}(i) - \mathbf{P}_{a_1}(i)\right)\left(\mathbf{I} - \gamma \mathbf{P}_{a_1}\right)^{-1}\right\} - \lambda \left\|\mathbf{R}\right\|_1, \tag{2.9}$$

$$\text{s.t.} \left(\mathbf{P}_{a_1} - \mathbf{P}_a\right)\left(\mathbf{I} - \gamma \mathbf{P}_{a_1}\right)^{-1} \mathbb{R} \succeq 0\, \forall a \in A \setminus a_1,\ |\mathbf{R}_i| \leq R_{\max}, \tag{2.10}$$

where $\mathbf{P}_{a_1}(i)$ denotes the ith row of $\mathbf{P}_{a_1}$. The λ is a adjustable penalty coefficient balancing between the goal of minimizing objective function 2.9 (large λ value), and avoiding reward values being zeros (small λ value). The objective function is the maximized difference between the value function of the optimal policy and the value function of sub-optimal policy.

This can be interpreted as the optimal reward function should be able to discern the optimal policy from others, in other words, optimal reward function maximizes the difference between the best policy and the next-best policy.

When dealing with large state space MDPs, IRL approximate the reward function using basis functions $\phi_i(s)$ mapping states to rewards., the new reward function estimate is

$$\begin{aligned} R(s) &= w_1\phi_1(s) + w_2\phi_2(s) + ... + w_d\phi_d(s) \\ &= \mathbf{w}\cdot\boldsymbol{\phi}(s). \end{aligned} \tag{2.11}$$

The reward function is the linear combination of some basis functions. Because the value function V is the expectation of rewards and the linearity of expectation, the value function can also be written as the linear combination of the same basis functions. Let V_i^π denotes the value function of policy π when the reward function is $R = \phi_i$, the value function is therefore [37]

$$V^\pi = w_1 V_1^\pi + ... + w_d V_d^\pi. \tag{2.12}$$

The value function in this form can be used to approximate the reward function using sampled demonstrator's trajectories, this extension leads to Apprenticeship Learning (AL) [38], a sub-type of IRL.

2.8.2. Apprenticeship Learning through Inverse Reinforcement Learning

Apprenticeship Learning through Inverse Reinforcement Learning (AL) deals with the same problem as dealt with by IRL. It can be seen as a method within the IRL category. The AL does not require a transition model like many IRL methods do, and the AL estimate the optimal reward function with performance measurement of policies called feature expectations, denote by $\mu(\pi)$. The feature expectation can be written as the expectation of the accumulative discounted reward when $R = w\cdot\phi$. Let τ denotes each possible trajectory in the environment and $\mathrm{P}(\tau|\pi)$ denotes the probability of trajectory τ when using policy π, the feature expectation of policy π is:

$$\mu(\pi) = E\left[\sum_{t=0}^{\infty}\gamma^t\phi(s_t)|\pi\right] = \sum_{\tau}\mathrm{P}(\tau|\pi)\sum_{t=0}^{\infty}\gamma^t\phi(s_t). \tag{2.13}$$

The expectation of feature expectation is the measurement of a policy, but the feature expectation of a trajectory τ does not contains information of the policy. Observe equation 2.13, only the term $\mathrm{P}(\tau|\pi)$ is related to π, in other words, the feature expectation of a policy can measure this policy because it contains the distribution of trajectories when using the policy. Similar to the problem setting in IRL, the problem for the AL is to find a policy which acts similarly to the demonstrator's policy, given the demonstrator's feature expectation vector μ_E. With a set D of m trajectories $\{s_0^i, s_1^i, ...\}_{i=1}^m$, the empirical estimate of μ_E is

$$\hat{\mu}_E = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{\infty}\gamma^t\phi\left(s_t^{(i)}\right). \tag{2.14}$$

The algorithm of AL is shown in the algorithm below.

The method used by this algorithm to compute $w^{(i)}$ is max-margin algorithms. Both Support Vector Machine (SVM) [39] and projection method are suitable for computing $w^{(i)}$.

Algorithm 5 Apprenticeship Learning via Inverse Reinforcement Learning[38]

Estimate demonstrator's feature μ_e expectation vector from a set of sampled demonstrator's trajectories D
Randomly initialize the first policy $\pi^{(0)}$, estimate its feature expectation vector $\mu^{(0)} = \mu\left(\pi^{(0)}\right)$, and set $i = 1$.
while Termination condition is not met **do**
  Compute $t^{(i)} = \max_{w:|w|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^\intercal \left(\mu_E - \mu^{(j)}\right)$, and let $w^{(i)}$ be the value that attains this maximum.
  if $t^{(i)} \leq \epsilon$ **then**
    Terminate the algorithm.
  end if
  Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using reward function $R(s) = \mathbf{w} \cdot \boldsymbol{\phi}(s)$.
  Set $i = i + 1$
end while

The SVM method label the demonstrator's feature expectation vector with 1 and the generated trajectories with -1, then SVM searches for the best hyperplane $w^{(i)}$ that separates data points with the maximum margin. The projection method, on the other hand, iterative searches for better $w^{(i)}$. A simple illustration of the projection method is shown in Figure 2.11.

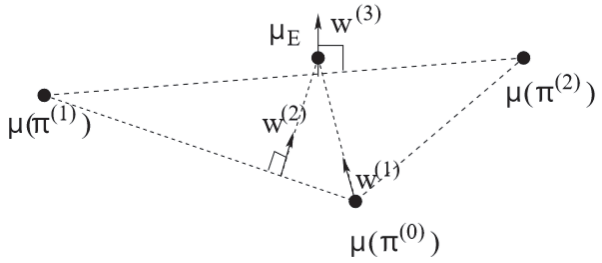


Figure 2.11: An illustration of projection method: $\mu\left(\pi^i\right)$ approaches to μ_E

AL has been used for helicopter acrobatics experiment [40]. It is shown in the experiment that using a combination of AL algorithm described above, and a generative model, a helicopter can learn to follow several difficult acrobatics trajectories demonstrated by a human operator. Snapshots of the experiment are shown in figure 2.12, the full video can be found in `http://heli.stanford.edu/`. It should be noted that in this work, AL does not directly produce actual actions, instead, the policy AL generated is used to deal with the uncertainty in the generative model, to improve its accuracy.

In the AL algorithm, the solution is ambiguous, many reward functions can be the solution to the demonstrated policy, and many policies have similar feature expectations [42]. The core of this ambiguity is that although the distribution of trajectories describe the policy, AL uses a simple and empirical approach to estimate it. Moreover, it is often an assumption that demonstrated trajectories are optimal. This assumption does not harm the convergence property of the AL algorithm [38], however it affects the quality of policies that

Figure 2.12: Snapshots of the autonomous helicopter acrobatics [41]

learned from the demonstration.

2.8.3. Maximum Entropy Inverse Reinforcement Learning

In the original AL algorithms, it is ambiguous to determine the matching feature expectation because for the feature expectation of a demonstrator, there usually exist many policies that produce trajectories with the same feature expectation. Maximum Entropy Inverse Reinforcement Learning (MaxEntIRL) [42] employs the principle of maximum entropy to address this problem. In MaxEntIRL, policies are modeled as a distribution over a set consists of all possible trajectories τ_i, $0 < i \leq N$. For each policy, the feature expectation of it can be written as

$$\mu_\pi = \sum_i P(\tau_i)\,\mu_{\tau_i} \tag{2.15}$$

where $P(\tau_i)$ is the probability distribution of trajectory τ_i in this policy, and μ_{τ_i} is the feature expectation of trajectory τ_i. Applying the maximum entropy principle, the distribution $P(\tau_i)$ should be modeled in such way that this distribution has the largest entropy in the context of given μ_π. The AL problems state that, given a measurement of the demonstrator's policy μ_E, a reward function parameterized by weights w and a feature vector $\phi(s,a)$ should be found. In MaxEntIRL, the derived policy distribution using maximum entropy principle can be written as [42]

$$P(\tau_i|w) = \frac{1}{Z(w)} e^{w^\top \mu_{\tau_i}} = \frac{1}{Z(w)} e^{\sum_{s_j,a_j \in \tau_i} w^\top \phi(s_j,a_j)} \tag{2.16}$$

where the partition function $Z(w) = \sum_{\tau_i} e^{w^\top \mu_{\tau_i}}$. Maximum Likelihood Estimation (MLE) and Stochastic Gradient Descent can be used to estimate the parameter w in distribution $P(\tau_i|w)$. The key to solve this optimization problem is finding the partition function $Z(w)$. MaxEntIRL assumes the dynamics of the environment is known, and the partition function can be easily calculated from the transition probability [42].

2.8.4. Relative Entropy Inverse Reinforcement Learning

In the original MaxEntIRL method, the partition function is calculated assuming known dynamics. But this proposal focuses on the model-free approaches. To combat this problem, Relative Entropy Inverse Reinforcement Learning (RelEntIRL) [43] employs relative entropy rather that maximum entropy in the original MaxEntIRL algorithm. The new entropy can be wirtten as

$$\max_P \sum_{\tau_i} P(\tau_i|w) \ln \frac{Q(\tau_i)}{P_(\tau_i|w)}. \tag{2.17}$$

$P(\tau_i|w)$ denotes the trajectory distribution of the policy found with a reward function linear in features, constructed with parameter w. $Q(\tau_i)$ is the trajectory distribution of the baseline policy, in other words, the policy of demonstrations. The new probability distribution $P(\tau_i|w)$ after solving this equation is [43]:

$$P(\tau_i|w) = \frac{1}{Z(w)} Q(\tau_i) e^{w^\top \mu_{\tau_i}} = \frac{1}{Z(w)} Q(\tau_i) e^{\sum_{s_j,a_j \in \tau_i} w^\top \phi(s_j,a_j)} \tag{2.18}$$

with $Z(w) = \sum_{\tau_i} Q(\tau_i) e^{w^\top \mu_{\tau_i}}$. $Q(\tau_i)$ can be decompose as $Q(\tau_i) = D(\tau_i)U(\tau_i)$ where $D(\tau_i) = D_0 \prod_{t=1}^{T} T(s_t, a_t, s_{t+1})$ is the joint probability of state transitions in τ_i, and $U(\tau_i)$ is the joint probability of the actions conditioned on states in τ_i [43]. The probability distribution becomes

$$P(\tau_i|w) = \frac{D(\tau_i)U(\tau_i)e^{w^\top \mu_{\tau_i}}}{\sum_{\tau_i} D(\tau_i)U(\tau_i)e^{w^\top \mu_{\tau_i}}} \tag{2.19}$$

By doing so, the only term that involve dynamics in this equation is $D(\tau_i)$. When solving the MLE problem of this distribution function, importance sampling ratio can be $D(\tau_i)\pi(\tau_i)$, so the dynamics $D(\tau_i)$ will be cancelled and the necessity for the transition model will disappear.

RelEntIRL is a sample-based approach to LfD problems, and it provides an efficient way of estimating the partition function in the original MaxEntrIRL problem. Other approaches can sample these trajectories more efficiently. For example, trajectories in the vicinity of demonstrations can be sampled more often with supervised learning [44], or the sample distribution can be refined over time using approximated local dynamics [45]. But because supervised learning and model-based methods are out of the scope, these methods will not be selected in this project.

2.9. Conclusion of Literature Review

This literature review is to collect the Reinforcement Learning (RL) methods and the Learning from Demonstrations (LfD) techniques in order to present the state-of-the-art approaches for flight control. The first part of this literature review shows the basics of RL and some of the existing implementation of RL algorithms for flight control. The second part of this literature review starts with a general description of LfD and then describe three strategies which can be used in LfD algorithms.

The Temporal Difference methods, especially Q-learning, are emphasized because of their nature of being able to update estimate without waiting for episodes finished, and being able to bootstrap estimate without interacting with the real environment. These methods are then extended to RL methods with approximation, specifically Semi-gradient methods with linear and non-linear features. Using these methods means at least one value function has to be estimated, which directly motivates some of the LfD strategies and the model learning procedures. After that, it is shown that policy-gradient methods parameterize policy and select actions from the estimated policy without consulting the value function. The policy-gradient methods link actions, or distributions of actions to states. Mapping is a LfD strategy that employs very similar idea to policy-gradient methods, it approximates the underlying function mapping from states to actions. Recent research shows Trust Re-

gion Policy Optimization, Proximal Policy Optimization and Deterministic Policy Gradient (DPG) methods enable efficient and robust learning for continuous state and action space.

For flight control, both model-based and model-free RL algorithms can be utilized. But he model-free approach is more attractive since the model of dynamics is generally difficult to obtain. The most common control task to test RL algorithms for flight control is tracking control, including position and speed tracking. The quadrotor is a common but representative platform, hence, the quadrotor is selected for LfD implementation. In addition, to facilitate learning and analysis on the result, the dynamics of aircraft can often be simplified into a one-dimensional model.

In general, LfD has three distinctive strategies, namely, mapping, planning, and model learning. It is shown in the literature that some mapping methods could not work with RL because these methods are usually task-specific and often lack the ability to explore the state space. A better mapping method with RL is called Deep Q from Demonstrations (DQfD), which approximates mapping function using DQN, a convolutional neural network. The use of demonstration in DQfD facilitates the initialization of DQN parameters and thus increases the learning efficiency. Literature about planning, the second LfD strategy, shows that this method does not typically work with RL, as it uses task-specific demonstrations consists of fixed, sequential state-action pairs with pre- and post-conditions. The third LfD strategy is model learning. Because the model learning uncover the demonstrator's intention by constructing the reward function, and the transition model of the environment, it is best suited for RL. Since the structure of RL includes autonomous explorations, the transition model can be directly learned by the agent without additional steps. So, a wide range of literature focuses on how to build the reward function. IRL is a well-developed approach towards the reward function reconstruction. IRL's variant, AL, typically deals with large state space problems using a performance measurement vector: feature expectations. MaxEntIRL and RelEntIRL modify the original AL algorithm to reduce the effect of feature expectation ambiguity.

The first part of the literature review answers the first research question and partially answers the second research question. The research question 1(a) is answered by the first part of literature review. For discrete state and action space, tabular methods such as tabular Q-learning are suitable. For continuous state and action space, methods with function approximator such as DDPG, TRPO and PPO are suitable. The research question 1(b) is also answered in the first part of literature review. The suitable flight control problem is a tracking control of a quadrotor in simulation. But for the convenience of analysis, this quadrotor can be simplified into a 1-D dynamics model. Hence the answer to the first research question is: tabular RL algorithms such as Q-learning are well suited for simplified flight control, specifically the control of a 1-D quadrotor for tracking tasks. Based on this answer and the analysis in the second part of literature review, the research question 2(a) can be answered. DQfD, IRL, AL, MaxEntIRL, RelEntIRL are methods in LfD. Among these methods, the IRL and its variants are best suited for further analysis because the model-learning strategy is suitable for incorporating RL algorithms.

3

Preliminary Analysis

The goal of this chapter is to show the effect reward functions built by LfD algorithms on RL algorithms in terms of sample efficiency, in order to answer the second and the third research question, while focusing particularly on two methods: IRL and AL. Specifically, the IRL algorithm and the AL algorithm will be implemented in a corridor simulation in order to show the effect of reward function they build. For comparison, the tabular Q-learning will be implemented with a handcrafted reward function. Different reward functions and action-value functions will be compared in heat-maps. The effect of these reward functions on RL algorithms will be shown by the learning curves.

3.1. Simulation Setup

This simulation is based on a corridor problem shown in figure 3.1, where the agent wants to pick up a cargo (green circle), and then drop it at the designated position (yellow block), and blue blocks represent other positions in the corridor. The total length of this corridor is 10, and the cargo is put at the first block. There are four actions available for the agent in any state, ranging from number 1 to 4, their effects are shown in the table 3.1. Taking any action equally consumes one time step. The action 3 does not have any effect when the agent already had the cargo, or the agent is not in the block where the cargo is located. After taking action 4, the episode will be terminated no matter the agent's location, but the episode where the agent drops the cargo in the correct block will be considered as a successful episode.

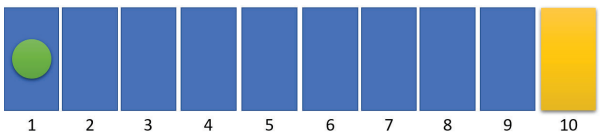


Figure 3.1: An illustration of corridor simulation setup.

At each time step, there are two states observable, one is the agent location defined as a number in set {1, 2, ..., 10}, the other is a flag indicating whether the agent has the cargo or

Table 3.1: 4 actions for the corridor simulation

Action number	Effects
1	Move one block forward, stay in block 10 when in block 10.
2	Move one block backward, stay in block 1 when in block 1.
3	Pick up the cargo at the current block
4	Drop the cargo at the current block. Terminate the episode.

not. Thus, the state space can be illustrated by figure 3.2. In the figure, states in the row above show the agent has the cargo while states in the row below show the cargo is still in the block. The goal state is defined as taking action 4 at block 10 when the agent has the cargo (highlighted as a yellow block).

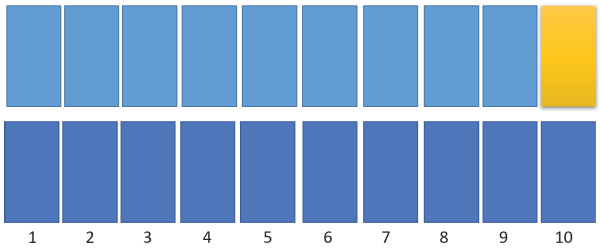


Figure 3.2: State space of the corridor simulation. First row: states where agent has the cargo, second row: states where the cargo is not picked up.

3.2. Implemented Algorithm

3.2.1. Q-learning

The first algorithm implemented is the tabular Q-learning introduced in section 2.2. The Q-learning algorithm is the benchmark for many RL algorithm implementation and thus it is used here to learn the optimal policy and test the reward function recovered by IRL algorithms. The details of tabular Q-learning algorithm is shown in the algorithm box 2, the policy used for action selections is ϵ-greedy policy, rewards are also discounted by factor γ to compute update target $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$. The value of γ is 0.999, and the value of ϵ is 0.9 to ensure that the agent explores all states. The default reward function for Q-learning gives the same reward for all state-action pairs except for special state-action pairs shown in reward function 3.1. The reward function is formalized as

$$R(s,a) = \begin{cases} R_1, & \text{if } (s,a) \text{ is the goal state} \\ R_2, & \text{if the agent pick up the cargo} \\ R_3, & \text{if the agent drop the cargo at a wrong location} \\ R_4, & \text{any other state-action pairs} \end{cases} \tag{3.1}$$

To compare the rate of successful episodes of different settings, different values of $\{R_1, R_2, R_3, R_4\}$ are used, they are shown in table 3.2

3.2.2. Inverse Reinforcement Learning

The second algorithm implemented is Inverse Reinforcement Learning for the discretized state, details of the algorithm can be found in section 2.8. IRL searches the reward function

Table 3.2: 4 reward settings for the corridor simulation

Setting number	R_1	R_2	R_3	R_4
1	10	1	−1	0
2	10	5	−1	0
3	10	10	−1	0
4	10	10	−10	0

$R(s)$ that describe demonstrator's behaviors D by optimizing the objective function:

$$\text{maximize} \sum_{i=1}^{N} \min_{a\in\{a_2,\dots a_k\}} \left\{ \left(\mathbf{P}_{a_1}(i) - \mathbf{P}_a(i)\right) \left(\mathbf{I} - \gamma \mathbf{P}_{a_1}\right)^{-1} \right\} - \lambda \|\mathbf{R}\|_1 \tag{3.2}$$

With conditions

$$\text{s.t. } \left(\mathbf{P}_{a_1} - \mathbf{P}_a\right) \left(\mathbf{I} - \gamma \mathbf{P}_{a_1}\right)^{-1} \mathbb{R} \succeq 0,\ \forall a \in A \setminus a_1,\ |\mathbf{R}_i| \leq R_{\max}. \tag{3.3}$$

In the objective function 3.2, the reward function $R(s)$ has been transformed into a vector $\mathbf{R}$, and $\mathbf{P}_{a_1}(i)$ denotes the ith row of transition probability matrices $\mathbf{P}_{a_1}$.

The first step of solving IRL problem is finding the probability matrix $\mathbf{P}_{a_1}$. This probability matrix contains probabilities of state transitions upon taking the optimal action a_1, so, this matrix can be estimated through Monte Carlo method using the optimal policy. Suppose it is available that a set D of m demonstrations $\{s_0^i, s_1^i, \dots s_T^i\}_{i=1}^m$. Let $\mathbf{P}_{a_1 t}^m$ denotes the transition probability matrix upon taking the optimal action at time step t in the mth episode, the empirical estimate of $\mathbf{P}_{a_1}$ is

$$\mathbf{P}_{a_1} = \frac{1}{m} \sum_{i=0}^{m} \sum_{t=0}^{T} \mathbf{P}_{a_1 t}^m. \tag{3.4}$$

It has been shown in the literature[42][46] that the original IRL, which is used here, assumes the optimality of demonstrations, sub-optimal demonstrations affect the value of information in reward function found by IRL algorithm. In this simulation, policy generated from Q-learning is used to approximate $\mathbf{P}_{a_1}$.

The next step of solving IRL problem is finding the probability matrix $\mathbf{P}_a$. In general, the objective function 3.2 optimized in IRL represents the difference between the expected return[1] of the demonstrator's policy (assumed the optimality) and the expected return of the sub-optimal policy. In this simulation, the expected return of the sub-optimal policy is easy to find, because only four actions are available in each state and one of these actions is used by the demonstrator's policy. Therefore, the expected return of the sub-optimal policy can be computed through Monte Carlo method using the policy that takes the sub-optimal action. With both $\mathbf{P}_{a_1}$ and $\mathbf{P}_a$, the objective function 3.2 can be solved by a linear programming method.

3.2.3. Apprenticeship Learning

The algorithm of AL is shown in section 2.8.2. In the AL, a reward function is the linear combination of features $\phi_i(s)$ as shown in equation 3.5. The goal is to find parameter $\mathbf{w}$ that

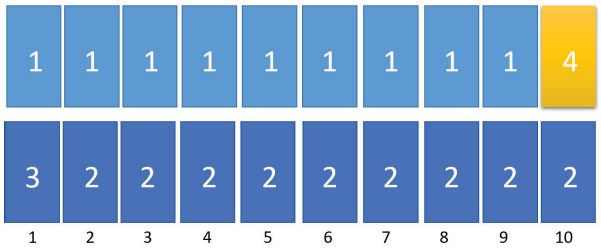


Figure 3.3: The optimal policy used to generate demonstrations.The number written on the each block is the number of action for this state.

best describe the demonstrator's behaviors by reconstructing the reward function $R(s,a)$

$$\begin{aligned} R(s,a) &= w_1\phi_1(s,a) + w_2\phi_2(s,a) + \ldots + w_d\phi_d(s,a) \\ &= w \cdot \phi(s,a) \end{aligned} \tag{3.5}$$

The feature vector $\phi(s)$ used in this simulation has 80 entries, each of them represent one state-action pair such that when that state-action pair is selected the corresponding element is 1 and the other elements remain 0.

To obtain the w, AL firstly measures the performance of a policy. This is done by estimating feature expectation $\mu(\pi)$, which is the expectation of the accumulated discounted return of the estimated policy:

$$\mu(\pi) = E\left[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi\right]. \tag{3.6}$$

It is assumed that the demonstrator uses the optimal policy π_E (shown by figure 3.3), then its feature expectation can be estimated by equation 2.14. Other policies can be measured in the same way.

Then, AL finds a better reward function parameter **w** that helps to match the agent's feature expectation estimate $\hat{\mu}$ to demonstrator's feature expectation μ_E, based on policies it has seen. Suppose the AL has already found i policies $\{\pi^1, \pi^2, \ldots \pi^i\}$ and their feature expectations are $\{\mu^1, \mu^2, \ldots \mu^i\}$, a parameter **w** can be found by SVM or the projection method according to section 2.8.2. In this way, the feature expectation of an agent's policy will gradually approach to the feature expectation of the demonstrator's policy. The AL terminated this searching when the two feature expectations are close enough.

3.3. Results

The first result is the action-value function of a ϵ-greedy policy estimated by Q-learning using the first reward setting in table 3.2. A heat map of this action-value function is shown in Figure 3.4. Higher action-values are highlighted by yellow color and lower action-values are blue. In Figure 3.4, the left sub-figure shows action-values when cargo is not obtained by the agent and the right sub-figure shows action value when the agent has obtained the cargo. In each sub-figure, the horizontal axis represents four actions and the vertical axis represents ten different blocks.

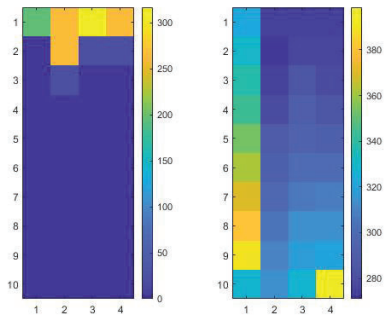


Figure 3.4: Action-value function heat-maps by Q-learning using reward function in 3.1

The reward function recovered by IRL is shown in Figure 3.5. This reward function has the same layout as the action-value function in Figure 3.4. Although this reward function identifies the goal state (bottom-left corner of the right figure), other states are not estimated accurately.

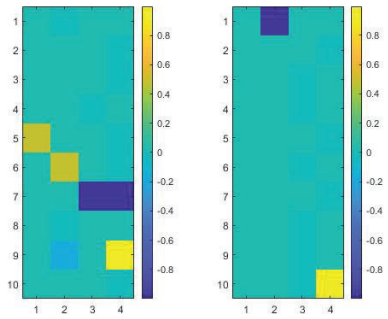


Figure 3.5: Reward function recovered by IRL

The reward function recovered by AL is shown Fn figure 3.6. In contrast to Figure 3.5, this reward function clearly identifies the goal state, as well as other state-action pairs by assigning them high rewards. For example, in the left sub-figure, the third block of the first row, which corresponds to *picking up the cargo*, is highlighted with a high reward. In the right sub-figure, the first column, except for the last row, corresponds to *moving the agent with cargo along the corridor until it reaches the terminal state*, the state-action pairs in this column are all highlighted with high rewards. Action-value function heat-map of a ϵ-greedy policy estimated by Q-learning using reward function in figure 3.6 is shown in Figure 3.7. Comparing Figure 3.4 and Figure 3.7, it can be seen that although magnitudes of action-value function are different, gradients in two figures are almost the same.

Finally, it is compared in figure 3.8 that the success rate of this simulation using reward settings in table 3.2 and the success rate using the reward function found by the AL. These reward functions are evaluated through Q-learning. The success rate with the reward function found by the AL is quickly converged to around 70%, while the success rates with other reward function gradually increase and approach to 70%.

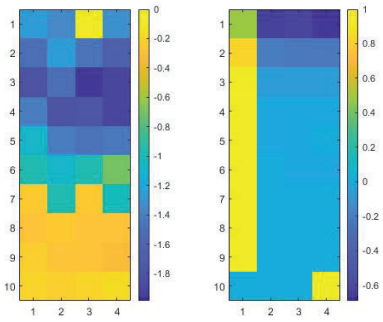


Figure 3.6: Reward function recovered by AL

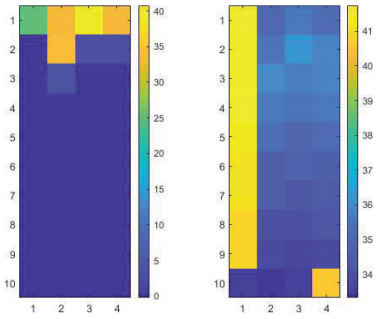


Figure 3.7: Action-value function heat map of a ϵ-greedy policy estimated by Q-learning using reward function in figure 3.6

3.4. Discussion

It has been shown the reward function found by AL is much more informative than the reward function found by IRL. One reason is IRL uses a transition matrix V^{π} using the optimal action, but AL uses feature expectations which is essentially an evaluation of policies. Although this transition matrix contains information about optimal actions, this information is not accurate because using some sub-optimal actions could produce the same transition matrix. Another reason is IRL algorithm is essentially built upon the assumption that actions for $\mathbf{P}_{a_1}$ are optimal. According to the paper [37], the condition $\left(\mathbf{P}_{a_1} - \mathbf{P}_a\right)\left(\mathbf{I} - \gamma \mathbf{P}_{a_1}\right)^{-1}\mathbb{R} \succeq 0$ is based on the value function $V^{\pi} = R + \gamma \mathbf{P}_{a_1} V^{\pi}$ when π given by $\pi \equiv a_1$ is optimal. Thus, when any action is not optimal, IRL can no longer find the correct solution to the reward function.

On the other hand, AL does not rely on the assumption of optimality as much as IRL does. The core idea of AL is to let the agent's feature expectation approach to demonstrator's feature expectation. If the demonstrator uses the optimal policy, the performance of AL is maximized, but if the demonstrator uses a sub-optimal policy, an informative reward function can still be found.

More importantly, AL does not require a transition matrix to rebuild reward functions. Because AL needs few RL processes to estimate the effect of rebuilt reward functions, while IRL does not need any estimate of the rebuilt reward functions.

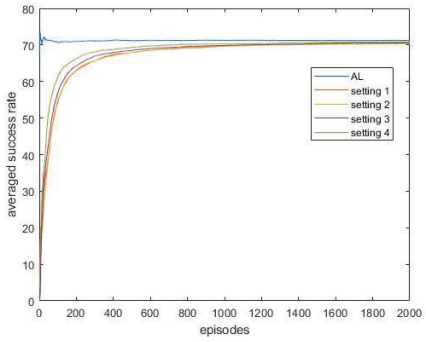


Figure 3.8: Averaged success rate using different reward functions.

3.5. Conclusion of Preliminary Analysis

In the preliminary analysis, simulations about a cargo-delivery task with Q-learning are conducted. Inverse Reinforcement Learning (IRL), and Apprenticeship Learning (AL) are used to construct reward functions from the demonstration provided by a human operator. The Q-learning is used to compare different handcrafted reward functions and the reward function built by algorithms. IRL and AL take different approaches to estimating the reward function that explains the demonstrator's behavior, but the reward function found by IRL is less informative. By comparing the success rate using different reward functions, it is found that AL generates a reward function that greatly facilitates the convergence of the Q-learning algorithm. This improvement is due to a sufficient amount of knowledge of the task that is transformed into reward signals. Furthermore, the IRL requires the transition model of environment, AL, however, is a model-free algorithm.

The preliminary analysis answers the second research question and the third research question. The sub-question 2(a) has been answered in the literature review. To answer sub-question 2(b), this section compares the effect of the IRL- and AL-constructed reward functions with the handcrafted reward functions. The simulation of corridor in this chapter has shown that although sophisticated reward functions are used, the regular Q-learning algorithms still cannot outperforms IRL and AL algorithms in terms of learning speed and success rate. Therefore IRL and AL algorithms help the RL to improve the sample efficiency by generating reward functions with demonstrations. To answer the third research question, the preliminary analysis presents the differences and similarity between the IRL and AL method. Based on the literature review, AL is built upon IRL, therefore both IRL and AL build reward functions using demonstration. Because AL does not rely on any model but can still produce similar reward function as the model-based IRL, AL is recognised as the most suitable LfD method for flight control in which an accurate model of environment is difficult to obtain.

4

Additional Results

This chapter serves as a supplementary document to the scientific paper in the first part of this thesis. In this chapter, meanings of λ in the original AL algorithm and κ in the PFP will be explained. The equations of feature expectation in SAL and AL will also be derived. Results and derivations in this chapter will support the experiment results that have been shown in Part 1.

4.1. λ and κ: Measurements of Feature Expectation

It has been shown in the scientific paper that variable λ equals variable κ which can be used to construct merged policies. Although the two variables share the same value, they have been given different meanings. The paper [38] mentioned that λ is a production of the original projection method with the relationship

$$\lambda_i = \frac{(\mu_i - \bar{\mu}_{i-1})^\top (\mu_D - \bar{\mu}_{i-1})}{(\mu_i - \bar{\mu}_{i-1})^\top (\mu_i - \bar{\mu}_{i-1})} \tag{4.1}$$

where the footnote i means the ith iteration in a run of the AL algorithm, μ_i is the feature expectation of the set of trajectories corresponding to the ith RL policy π_i. Then the relationship between μ_i and $\bar{\mu}_i$ can be established with

$$\bar{\mu}_i \;=\; \lambda_i \mu_i \;+\; (1-\lambda_i)\bar{\mu}_{i-1} \tag{4.2}$$

Assuming the feature expectation is two-dimensional, this relationship can be illustrated by Figure 4.1 in which $\bar{\mu}_i$ is located on the foot of perpendicular, p_1 is the length of vector $\mu_i - \bar{\mu}_i$ and p_2 is the length of vector $\bar{\mu}_i - \bar{\mu}_{i-1}$. Hence, the following relationships hold

$$l_1 + l_2 = (\mu_i - \bar{\mu}_{i-1})^\top (\mu_i - \bar{\mu}_{i-1}) \tag{4.3}$$

$$l_1 = (\mu_D - \mu_i)^\top (\mu_i - \bar{\mu}_{i-1}) \tag{4.4}$$

$$l_2 = (\mu_i - \bar{\mu}_{i-1})^\top (\mu_D - \bar{\mu}_{i-1}) \tag{4.5}$$

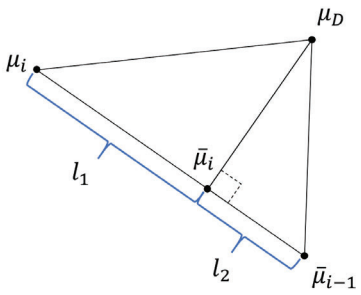
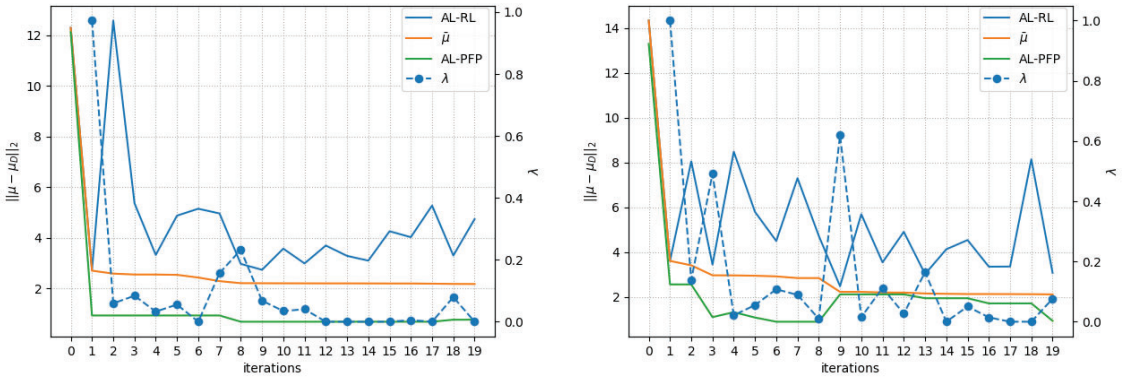


Figure 4.1: Progress of one iteration step, adapted from [38]

Thus, the variable λ can also be written as

$$\lambda_i = \frac{l_2}{l_1 + l_2} \tag{4.6}$$

The meaning of variable λ_i is the comparison of the distance from μ_i to μ_D with the distance from $\bar{mu}_{i-1}$ to μ_D. As μ_i gets closer to μ_D, the value of λ_i increases, and as $\bar{\mu}_{i-1}$ gets closer to μ_D, the value of λ_i decreases. Experiment results also shows the same relationship between λ_i, μ_i and $\bar{\mu}_i$. Figure 4.2 shows two representative runs of the AL algorithm using good and bad demonstration respectively. Distance of feature expectation μ to μ_D is measured by the Euclidean norm of vector $\mu - \mu_D$. This figure shows when μ_i is close to $\bar{mu}_i$, the value of λ_i increases.



(a) $\|\mu - \mu_D\|_2$ and λ, using bad demonstration

(b) $\|\mu - \mu_D\|_2$ and λ, using bad demonstration.

Figure 4.2: Comparison of feature expectation distances and λ in one run

However the variable κ does not have such features, because the feature expectation of demonstration is fixed in AL, but changing in SAL. It has been defined in Part 1 the expression of κ_i as follows

$$\kappa_i = \frac{(\mu_i - \bar{\mu}_{i-1})^\top (\mu_D - \bar{\mu}_{i-1})}{(\mu_i - \bar{\mu}_{i-1})^\top (\mu_i - \bar{\mu}_{i-1})} \tag{4.7}$$

For the SAL algorithm, μ_D is replaced by $\mu_{D(i)}$ to indicate the set of demonstrated trajectories is changing. Assuming $\bar{\mu}'_{i-1} = \bar{\mu}_{i-1}$, Figure 4.3 shows the change of $\mu_{D(i)}$ cause the relationships established in AL do not hold anymore. The following new relationships can be established

$$l'_1 = (\mu_{D(i+1)} - \mu_i)^\top (\mu_i - \bar{\mu}'_{i-1}) \tag{4.8}$$

$$l_2^{'} = (\mu_i - \bar{\mu}_{i-1}^{'})^{\mathsf{T}}(\mu_{D(i+1)} - \bar{\mu}_{i-1}^{'}) \tag{4.9}$$

$$\kappa_i = \frac{l_2^{'}}{l_1^{'} + l_2^{'}} \tag{4.10}$$

Therefore κ_i does not increase or decrease in the same way as the λ_i does, and when the set of demonstrated trajectories has changed, every κ_i has to be recomputed.

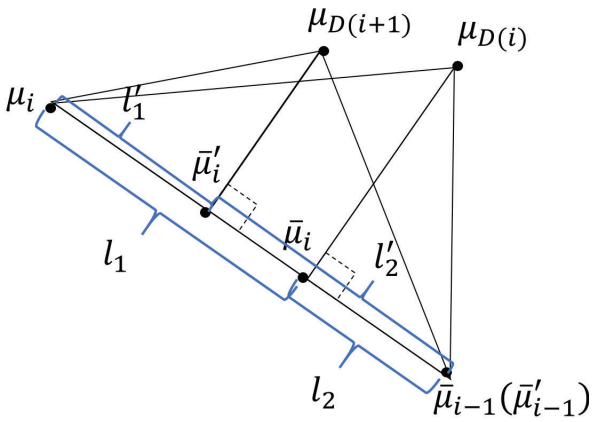


Figure 4.3: Progress of two iteration steps in the SAL, adapted from [38]

4.2. From λ and κ to Feature Expectations

The post-processing step in [4] uses the feature of λ to construct a distribution of policies, using the equation

$$\rho_i = \begin{cases} \pi_i, & p_i = \lambda_i \\ \pi_{i-1}, & p_{i-1} = (1-\lambda_i)\lambda_{i-1} \\ \vdots & \vdots \\ \pi_0, & p_0 = (1-\lambda_i)(1-\lambda_{i-1})...(1-\lambda_1) \end{cases} \tag{4.11}$$

Let $\mu(\rho_i)$ denotes the feature expectation of the set of trajectories when using distributed policy ρ_i. Using notations in Part 1, in the AL the following equation holds.

$$\mu(\rho_i) = \sum_{j=0}^{i} p_j \sum_{\tau \in \mathcal{T}_{env}} P(\tau|\pi_j) \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t|\tau) = \sum_{j=0}^{i} p_j \mu(\pi_j) = \lambda_i \mu_i + (1-\lambda_i)\bar{\mu}_{i-1} \tag{4.12}$$

Therefore the feature expectation of distributed policy ρ_i is $\bar{\mu}_i$ in AL. The second norm of $\bar{\mu}_i - \mu_D$ is monotonically decreasing over iteration steps in AL [38], so the distributed policy ρ_i should be able to produce feature expectation that is more and more similar to the demonstration's feature expectation.

However, the feature expectation of policy $\hat{\pi}_i$ in PFP has no such relationship to the feature expectations of past policies. Recall that policy $\hat{\pi}$ is estimated by

$$\hat{\pi}_i = \kappa_i \pi_i + (1-\kappa_i)\hat{\pi}_{i-1} \tag{4.13}$$

and the probability of taking trajectory τ with a policy π is

$$P(\tau|\pi) = P(s_0) \prod_{t=1}^{T} \pi(s_t, a_t) P_T(s_{t+1}|s_t, a_t, \tau) \tag{4.14}$$

So the feature expectation of $\hat{\pi}_i$ is

$$\begin{aligned} \mu(\hat{\pi}_i) = & \sum_{\tau \in \mathcal{T}_{env}} P(s_0) \prod_{t=1}^{T} \kappa_i \pi_i(s_t, a_t) P_T(s_{t+1}|s_t, a_t, \tau) \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t|\tau) \\ & + \sum_{\tau \in \mathcal{T}_{env}} P(s_0) \prod_{t=1}^{T} (1-\kappa_i) \hat{\pi}_i(s_t, a_t) P_T(s_{t+1}|s_t, a_t, \tau) \sum_{t=0}^{T} \gamma^t \phi(s_t, a_t|\tau) \end{aligned} \tag{4.15}$$

Therefore $\mu(\hat{\pi}_i) \neq \bar{\mu}_i$, and it is not clear the relationship between $\mu(\hat{\pi}_i)$ and $\bar{mu}_i$.

5

Conclusion

The objective of this research is to *improve Reinforcement Learning algorithms in terms of sample-efficiency, in the context of model-free Reinforcement Learning, by evaluating and improving the state-of-the-art Learning from Demonstrations methods that are suitable for flight control.* There are two sub-goals for this objective. They are:

1. Select and implement state-of-the-art model-free Reinforcement Learning and Learning from Demonstrations algorithms methods and observe the difference of performance, by making an evaluation about the suitability of these methods for the selected flight control simulation.

2. Improve Learning from Demonstrations methods by making an investigation on the advantages and disadvantages of existing Learning from Demonstrations methods as well as possible solutions.

This thesis contains a preliminary study around the first sub-goal and a scientific paper that achieves the second sub-goal. The preliminary study compares a broad range of state-of-the-art RL and LfD algorithms by analyzing literature and conducting preliminary experiments, to select the LfD method that is suitable for flight control while incorporating RL. Research questions and their sub-questions are answered in the preliminary study and the scientific paper. These questions are

1. Which state-of-the-art model-free Reinforcement Learning methods have the potential to be implemented in flight control?

   (a) What are the state-of-the-art model-free Reinforcement Learning methods for discrete and continuous problems?

   (b) What are the flight control problems to be solved?

2. What is the effect of incorporating the reward functions produced by existing Learn-

ing from Demonstrations methods into Reinforcement Learning algorithms?

   (a) What are the Learning from Demonstrations methods?

   (b) What are the effects of the reward functions given by Learning from Demonstrations?

3. Which Learning from Demonstrations methods is suitable for flight control?

4. How to improve Learning from Demonstrations methods?

   (a) What are the disadvantages of using the existing Learning from Demonstration methods?

   (b) What are the improvements can be made on these disadvantages?

The first research question and a part of the second research question are answered by Chapter 2. The research question 1(a) is answered by the first part of the literature review which includes an introduction of basic RL components and a brief comparison between some of the state-of-the-art RL methods. The answer is: for small, discrete state and action space, tabular methods such as tabular Q-learning are suitable; for large, continuous state and action space, methods with function approximators such as DDPG, TRPO, and PPO are suitable. To answer the research question 1(b), existing implementations of RL in flight control are introduced. The answer to this question is a tracking task with simplified 1-D quadrotor dynamics. Therefore the answer to the first research question is: tabular RL algorithms such as Q-learning are well suited for simplified flight control, specifically the control of 1-D quadrotor for tracking tasks.

Based on the analysis in section 2.8, research question 2(a) can be answered. DQfD, IRL, AL, MaxEntIRL, RelEntIRL are the most representative LfD methods. Among these methods, IRL and its variants are best suited for further analysis as it adopts the strategy of model-learning which is well-connected to RL algorithms.

Chapter 3 answers both the second and the third research questions by comparing the effects of the IRL- and AL-constructed reward functions with the handcrafted reward functions. The simulation of the corridor has shown that the tabular Q-learning algorithms with handcrafted reward functions cannot outperform the IRL and AL algorithms in terms of sample efficiency. Therefore the answer to research question 2(b) is: the IRL and AL algorithm can help RL to improve the sample efficiency by generating reward functions using demonstrations. To answer the third research question, Chapter 3 focuses on the differences between the IRL and AL method. Both IRL and AL build a reward function with provided demonstrations. However, AL does not rely on any model but can still produce a similar reward function as the model-based IRL. AL is thus recognized as the most suitable LfD method for flight control.

The last research question is answered by the scientific paper. The scientific paper aims to

improve the AL and related methods. Existing algorithms such as the original AL through IRL, MaxEntIRL, RelEntIRL cannot learn policies that surpass the performance of the provided demonstration. The reason is that these algorithms cannot distinguish and replace underperforming trajectories with trajectories that have better performance. They cannot efficiently produce policies that imitate the demonstrator. This is caused by the fact that AL algorithms use only the reward function as the output, and an extra RL training step is required to translated the reward function into a policy. To deal with these disadvantages, improvements have been made on the original AL algorithm. The first improvement is the PFP method which merges all historic policies into a reliable one by using a majority voting mechanism. The second improvement is the trajectory replacing mechanism which ensure underperforming trajectories in the demonstration can be replaced. These two improvements are summarized by the SAL algorithm as a model-free AL algorithm based on the PFP method.

SAL has been compared with AL and the tabular Q-learning in the scientific paper. The simulation on a quadrotor control task shows AL cannot surpass the demonstration. It performs almost as good and also as bad as the demonstration. Trained with the same amount of samples, Q-learning algorithm using a handcrafted reward performs better than AL with the bad demonstration, but worse than AL with the good demonstration. When the bad demonstration is provide, SAL surpasses the performance of both bad demonstration and Q-learning. It also performs almost as good as the good demonstration. When the demonstration provided to SAL has poor performance, the algorithm tends to take a large amount of trajectories as the replacement of the trajectories in the original demonstration; when the provided demonstration performs well, a small amount of trajectories in the original demonstration is replaced. It should be noticed that the SAL method exhibits the capability of generalising policies from trajectories that may contain the noise introduced by RL. However, it is not clear that to what extent can SAL reduce the effect of noise.

This research has achieved the objective. Existing algorithms that use demonstrations has been reviewed, AL is thus selected as the suitable method for flight control. Being a model-free LfD method based on AL, SAL improves the sample-efficiency and presents an approach to surpassing demonstration's performance. For future research, it would be interesting to develop SAL with continuous actions. Because discrete actions fail to meet the requirements for many real quadrotor control tasks. State and action representations in SAL could also be extended with function approximators such as Neural Networks. Theories should also be established in the future to explain the phenomenon that SAL and PFP could produce policies that have very similar feature expectations to the demonstrator's feature expectation. For the SAL algorithm, it would be interesting to tune the size of the buffer that stores demonstration. Different performance measurements and different demonstrations could be used in SAL. The SAL method has the potential to be utilized for flight control in real life. The implementation of SAL will influence the development of flight control as well as the development of RL and LfD algorithms.

Bibliography

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[3] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 4565–4573.

[4] Umar Syed and Robert E Schapire. "A game-theoretic approach to apprenticeship learning". In: *Advances in neural information processing systems*. 2008, pp. 1449–1456.

[5] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[6] Christopher John Cornish Hellaby Watkins. "Learning from delayed rewards". In: (1989).

[7] Satinder P Singh and Richard S Sutton. "Reinforcement learning with replacing eligibility traces". In: *Machine learning* 22.1-3 (1996), pp. 123–158.

[8] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[9] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[10] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[11] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[12] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Thirtieth AAAI conference on artificial intelligence*. 2016.

[13] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[14] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), p. 484.

[15] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[16] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[17] Thomas Degris, Martha White, and Richard S Sutton. “Off-policy actor-critic”. In: *arXiv preprint arXiv:1205.4839* (2012).

[18] John Schulman et al. “Trust region policy optimization”. In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[19] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[20] David Silver et al. “Deterministic policy gradient algorithms”. In: *ICML*. 2014.

[21] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *arXiv preprint arXiv:1509.02971* (2015).

[22] Vijay R Konda and John N Tsitsiklis. “Actor-critic algorithms”. In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[23] Silvia Ferrari and Robert F Stengel. “Online adaptive critic flight control”. In: *Journal of Guidance, Control, and Dynamics* 27.5 (2004), pp. 777–786.

[24] Steven Lake Waslander et al. “Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning”. In: *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2005, pp. 3712–3717.

[25] Jemin Hwangbo et al. “Control of a quadrotor with reinforcement learning”. In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2096–2103.

[26] Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. “Locally weighted learning”. In: *Lazy learning*. Springer, 1997, pp. 11–73.

[27] Sonia Chernova and Manuela Veloso. “Confidence-based policy learning from demonstration using gaussian mixture models”. In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM. 2007, p. 233.

[28] Paul E Rybski and Richard M Voyles. “Interactive task training of a mobile robot through human gesture recognition”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 1. IEEE. 1999, pp. 664–669.

[29] Claude Sammut et al. “Learning to fly”. In: *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 385–393.

[30] Todd Hester et al. “Deep q-learning from demonstrations”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[32] Yan Duan et al. “One-shot imitation learning”. In: *Advances in neural information processing systems*. 2017, pp. 1087–1098.

[33] William Koch et al. “Reinforcement learning for UAV attitude control”. In: *ACM Transactions on Cyber-Physical Systems* 3.2 (2019), p. 22.

[34] Harini Veeraraghavan and Manuela Veloso. “Teaching sequential tasks with repetition through demonstration”. In: *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems. 2008, pp. 1357–1360.

[35] Andrew Garland and Neal Lesh. “Learning hierarchical task models by demonstration”. In: *Mitsubishi Electric Research Laboratory (MERL), USA–(January 2002)* (2003).

[36] Stuart J Russell. “Learning agents for uncertain environments”. In: *COLT*. Vol. 98. 1998, pp. 101–103.

[37] Andrew Y Ng, Stuart J Russell, et al. “Algorithms for inverse reinforcement learning.” In: *Icml*. Vol. 1. 2000, p. 2.

[38] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 1.

[39] Johan AK Suykens and Joos Vandewalle. “Least squares support vector machine classifiers”. In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[40] Pieter Abbeel. *Apprenticeship learning and reinforcement learning with application to robotic control*. Stanford University, 2008.

[41] Tse Ben and Fratkin Eugene. *High-resolution photos for the autonomous airshow*. 2008. URL: `http://heli.stanford.edu/photos.html`.

[42] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[43] Abdeslam Boularias, Jens Kober, and Jan Peters. “Relative entropy inverse reinforcement learning”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 182–189.

[44] Yevgen Chebotar et al. “Path integral guided policy search”. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3381–3388.

[45] Chelsea Finn, Sergey Levine, and Pieter Abbeel. “Guided cost learning: Deep inverse optimal control via policy optimization”. In: *International Conference on Machine Learning*. 2016, pp. 49–58.

[46] Dylan Hadfield-Menell et al. “Cooperative inverse reinforcement learning”. In: *Advances in neural information processing systems*. 2016, pp. 3909–3917.