HUMAN-IN-THE-LOOP METHODS FOR DATA-DRIVEN AND REINFORCEMENT LEARNING SYSTEMS

A Dissertation

by

VINICIUS GUIMARAES GOECKS

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

DOCTOR OF PHILOSOPHY

Chair of Committee,	John Valasek
Committee Members,	Gregory Chamitoff
	Daniel Selva
	Dylan Shell
Head of Department,	Rodney D. W. Bowersox

May 2020

Major Subject: Aerospace Engineering

ABSTRACT

Recent successes combine reinforcement learning algorithms and deep neural networks, despite reinforcement learning not being widely applied to robotics and real world scenarios. This can be attributed to the fact that current state-of-the-art, end-to-end reinforcement learning approaches still requires thousands or millions of data samples to converge to a satisfactory policy and are subject to catastrophic failures during training. Conversely, in real world scenarios and after just a few data samples, humans are able to either provide demonstrations of the task, intervene to prevent catastrophic actions, or simply evaluate if the policy is performing correctly. This research investigates how to integrate these human interaction modalities to the reinforcement learning loop, increasing sample efficiency and enabling real-time reinforcement learning in robotics and real world scenarios. The theoretical foundation of this research work builds upon the actor-critic reinforcement learning architecture, the use of function approximation to represent action- and value-based functions, and the integration of different human interaction modalities; namely, task demonstration, intervention, and evaluation, to these functions and to reward signals. This novel theoretical foundation is called *Cycle-of-Learning*, a reference to how different human interaction modalities are cycled and combined to reinforcement learning algorithms. This approach is validated on an Unmanned Air System (UAS) collision avoidance and landing scenario using a high-fidelity simulated environment and several continuous control tasks standardized to benchmark reinforcement learning algorithms. Results presented in this work show that the reward signal that is learned based upon human interaction accelerates the rate of learning of reinforcement learning algorithms, when compared to traditional handcrafted or binary reward signals returned by the environment. Results also show that learning from a combination of human demonstrations and interventions is faster and more sample efficient when compared to traditional supervised learning algorithms. Finally, *Cycle-of-Learning* develops an effective transition between policies learned using human demonstrations and interventions to reinforcement learning. It learns faster and uses fewer interactions with the environment when compared to state-of-the-art algorithms.

The theoretical foundation developed by this research opens new research paths to human-agent teaming scenarios where autonomous agents are able to learn from human teammates and adapt to mission performance metrics in real-time and in real world scenarios.

DEDICATION

To my wife, family, and friends who have always supported me throughout my academic journey.

ACKNOWLEDGMENTS

First of all, I would not be able to complete this doctorate degree without the support of my loving wife, Lucieni, and my parents, Elizabeth and Claudio, and brother, Thiago. My wife has been always by my side when facing the challenges of graduate school and also sharing all the happy moments that came with it. My parents, even though in a another country, were always supportive of me when pursuing this degree. I would not be able to complete this dissertation without their emotional support.

I would like to acknowledge and thank who contributed to the development of this research and the successful completion of this dissertation. Dr. John Valasek, who has been an exceptional advisor and mentor not only on academic matters, but also on leadership and management, which I see as invaluable skills for my future career. My committee members, Dr. Gregory Chamitoff, Dr. Dylan Shell, and Dr. Daniel Selva for their guidance when grounding the research fundamentals and support when defining the research direction. I would like to specially thank Dr. Nicholas Waytowich, Dr. Gregory Gremillion, and Dr. Vernon Lawhern, from the U.S. Army Research Laboratory, for being an integral part of the development process of this research, for working together when defining the research direction and research questions to be answered, and for all support when presenting and publishing the research results. This research would not be successful without their technical contributions, guidance, and support.

Finally, I would like to also thank my graduate student peers and personal friends that directly contributed to my formation and supported me during my doctorate journey: Humberto Ramos, Niladri Das, Hakjoo Kim, Jack Lu, Bochan Lee, Josh Harris, Robyn Woollands, Austin Probe, Clark Moody, and more recently, Ritwik Bera and Morgan Wood. Thank you for everything you taught me throughout this graduate school years and for being part of this journey.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Professor John Valasek, Gregory Chamitoff, and Daniel Selva of the Department of Aerospace Engineering and Professor Dylan Shell of the Department of Computer Science.

The data analyses for chapters 7 through 10 were conducted in collaboration with Dr. Nicholas R. Waytowich, Dr. Vernon J. Lawhern, and Dr. Gregory M. Gremillion from the U.S. Army Research Laboratory and were published in 2018 and 2019 in articles listed in the Biographical Sketch.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by a fellowship from Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, by a fellowship from the U.S. Army Research Laboratory through the Oak Ridge Associated Universities, and by a fellowship from the Department of Aerospace Engineering at Texas A&M University.

Research was sponsored by the U.S. Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-18-2-0134 and W911NF-17-2-0078. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

NOMENCLATURE

A3C	Asynchronous Advantage Actor-Critic
AAAI	Association for the Advancement of Artificial Intelligence
$A(s, a)$	Advantage function
a_t	Action vector at time t
AGL	Altitude Above Ground Level
AIAA	American Institute of Aeronautics and Astronautics
ANN	Artificial Neural Network
A-RLC	Adaptive-Reinforcement Learning Control
BCE	Binary Cross-Entropy
CE	Cross-Entropy
CNN	Convolutional Neural Network
CoL	Cycle-of-Learning for Autonomous Systems
BC	Behavior Cloning
CEM	Cross-Entropy Method
CMA-ES	Covariance-Matrix Adaptation Evolution Strategy
COACH	Convergent Actor-Critic by Humans
CPU	Central Processing Unit
D	Discriminator network
DAgger	Dataset Aggregation
DAPG	Demo-Augmented Policy Gradient
Deep RL	Deep Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient

DDPGfD	Deep Deterministic Policy Gradient from Demonstrations
DPG	Deterministic Policy Gradient
DL	Deep Learning
DQN	Deep Q-Network
DQfD	Deep Q-learning from Demonstrations
DOF	Degree-of-Freedom
ELU	Exponential Linear Unit
$J(\cdot)$	Objective function
HCRL	Human-Centered Reinforcement Learning
HER	Hindsight Experience Replay
HRI	Human-Robot Interaction
I2A	Imagination-Augmented Agents
IEEE	Institute of Electrical and Electronics Engineers
IJCNN	International Joint Conference on Neural Networks
IL	Imitation Learning
iLQR	Iterative Linear Quadratic Regulator
IMPALA	Importance Weighted Actor-Learner Architecture
IOC	Inverse Optimal Control
IPG	Interpolated Policy Gradient
IRL	Inverse Reinforcement Learning
IRM	Intrinsic Reward Module
FCN	Fully-Connected Network
FCCN	Fully-Connected Convolutional Network
G	Generator network
GAE	Generalized Advantage Estimator

GAIL	Generative Adversarial Imitaiton Learning
GAN	Generative Adversarial Network
GCL	Guided Cost Learning
GPS	Global Positioning System
GPU	Graphics Processing Unit
KL	Kullback-Leibler
$\mathcal{L}$	Loss function
LfD	Learning from Demonstrations
LfE	Learning from Evaluations
LfI	Learning from Interventions
LSTM	Long Short-Term Memory
LP	Linear Programming
M-PAC	Multi-Preference Actor Critic
MAE	Mean Absolute Error
MADCAT	Mission Adaptive Digital Composite Aerostructure Technologies
MDN	Mixture Density Network
MDP	Markov Decision Process
MLP	Multilayer Perceptron
MPC	Model Predictive Control
MSE	Mean Squared Error
MVE	Model Value Expansion
NAF	Normalized Advantage Function
o_t	Observation vector at time t
OML	Outer Mold Line
π	Policy

π_0	Initial Policy
PDMS	Polydimethylsiloxane
PER	Prioritized Experience Replay
PG	Policy Gradient
POfD	Policy Optimization with Demonstrations
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
$Q(s, a)$	State-action value function
$\mathcal{R}$	Replay Buffer
R_1	1-step return
R_n	n-step return
r_t	Reward value at time t
ReLU	Rectified Linear Unit
RGB	Visible spectrum color code
RL	Reinforcement Learning
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
s_t	State vector at time t
SAA	Sense-and-Avoid
SAC	Soft Actor-Critic
SAMI	Structured Adaptive Model Inversion
SC2LE	StarCraft II Learning Environment
SMA	Shape-Memory Alloy
sUAS	Small Unmanned Air System
SVM	Support Vector Machine

t	Time step
T	Total time steps in one episode
τ	Trajectory
TAMER	Training an Agent Manually via Evaluative Reinforcement
TAMU	Texas A&M University
Tanh	Hyperbolic Tangent
TD	Temporal Difference
TD3	Twin Delayed DDPG
θ_π	Parameters of policy π (actor)
θ_Q	Parameters of the critic
TNPG	Truncated Natural Policy Gradient
TRPO	Trust Region Policy Optimization
μ	Deterministic policy
UAS	Unmanned Air System
UAV	Unmanned Air Vehicle
UNREAL	Unsupervised Reinforcement and Auxiliary Learning
$V(s)$	State value function
VAE	Variational Autoencoder
VICE-RAQ	Variational Inverse Control with Events - Reinforcement learning with Active Queries
VPG	Vanilla Policy Gradient
VSCL	Vehicle Systems & Control Laboratory
w	Weight vector of a model
Z	Probability normalization term

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

"We call ourselves *Homo sapiens* — man the wise — because our intelligence is so important to us. For thousand of years, we have tried to understand how we think; that is, how a mere handful of matter can perceive, understand, predict, and manipulate a world far larger and more complicated than itself. The field of artificial intelligence, or AI, goes further still: it attempts not just to understand but also build intelligent entities."

— Stuart Russell. "*Artificial Intelligence: A Modern Approach*", 2003 [10].

1.1 Research Problem Overview

Data-driven approaches and learning algorithms are well suited to solve high-level prediction and control problems in an information-rich world. Learning algorithms are able to learn directly from examples, to search for an underlying pattern of an apparently patternless data, and to improve a decision-making process by continuously repeating it and observing the results.

The primary goal of learning methodologies is to imbue intelligent agents with the capability to autonomously and successfully perform complex tasks, when *a priori* design of the necessary behaviors is intractable. Instead, given an objective function for the desired behavior, learning techniques can be used to empirically discover the policy or controller required to satisfy it. Several classes of these techniques have yielded promising results, including reinforcement learning, learning from demonstrations, interventions, and evaluations.

Reinforcement learning has been shown to work on scenarios with well-designed reward functions and easily available interactions with the environment. However, in real-world robotic applications, explicit reward functions are non-existent, and interactions with the hardware are expensive and susceptible to catastrophic failures. This motivates leveraging human interaction to supply this reward function and task knowledge, to reduce the amount of high-risk interactions with the environment and to safely shape the behavior of robotic agents, thus, enabling *Real-Time Human-in-the-Loop Reinforcement Learning*.

1.2 Literature Review

This research focuses on two main areas: learning from human interaction and reinforcement learning.

1.2.1 Learning from Human Interaction

This section addresses the state-of-the-art and current challenges when learning from human interaction, including learning from human demonstrations, interventions, and evaluations. Other human interaction modalities considered but not included on this work are natural language, eye gaze tracking, gestures, brain electrical activity, and domain knowledge that is applied only to a single task.

1.2.1.1 Learning from Demonstrations

Learning from Demonstrations (LfD) can be used to provide a more directed path to these intended behaviors by utilizing examples of humans performing the task. This technique has the advantage of quickly converging to more stable behaviors. However, given that it is typically performed offline, it does not provide a mechanism for corrective or preventative inputs when the learned behavior results in undesirable or catastrophic outcomes, potentially due to unseen states. LfD also inherently requires the maximal burden on the human, requiring them to perform the task many times until the state space has been sufficiently explored, so as to generate a robust policy. Also, it necessarily fails when the human is incapable of performing the task successfully at all.

There are many empirical successes of using imitation learning to train a policy or controller based on human demonstrations [11]. Early research on learning from human demonstrations was primarily focused on teaching higher-level commands, as for example “pick", “move", and “place" when controlling a robotic arm [12, 13, 14], which later shifted to trajectory-level planning when the term “Learning from Demonstrations" became popular [15, 16, 17, 14].

For self-driving cars, the earlier Autonomous Land Vehicle In a Neural Network (ALVINN) [18] by Pomerleau learned from demonstrations to map from images to discrete actions using a single hidden-layer neural network. Most recent research also successfully used human demonstrations

to train a policy that mapped from front-facing camera images to steering wheel commands using around one hundred hours of human driving data [19]. Similar approaches have been taken to train small unmanned air system (sUAS) to navigate through cluttered environments while avoiding obstacles [20]. Nair et al. [21] combining self-supervised learning and behavior cloning used images as inputs to create a pixel-level inverse dynamics model of a robotic rope manipulation task. Related to navigation, there is related work that uses inverse optimal control on human demonstrations to learn navigation skill in a real world robot, and detect failure states and learn recover policies using Gaussian processes [22].

Rahmatizadeh et al. [23] demonstrated that humans can also aid robotic learning through demonstration of the goal task. Long Short-Term Memory (LSTM) networks [24] can be used to generalize human demonstration from virtual environments and have the learned policy transferred to a physical robot. When human data are limited, Deisenroth et al. [25] used Gaussian Processes to extract more information about each interaction between human and robot to reduce the time required to learn the robotic tasks. These approaches currently limit the robot behavior to what has been demonstrated by the human expert.

Another example of work that attempts to augment learning from demonstrations with additional human interaction is the Dataset Aggregation (DAgger) algorithm [26]. DAgger is an iterative algorithm that consists of two policies, a primary agent policy that is used for direct control of a system, and a reference policy that is used to generate additional labels to fine-tune the primary policy towards optimal behavior. Importantly, the reference policy's actions are not taken, but are instead aggregated and used as additional labels to re-train the primary policy for the next iteration. In [27] DAgger was used to train a collision avoidance policy for an autonomous quadrotor using imitation learning on a set of human demonstrations to learn the primary policy and using the human observer as a reference policy. There are some drawbacks to this approach that are worth discussing. As noted by Ross et al. [27], because the human observer is never in direct control of the policy, safety is not guaranteed, since the agent has the potential to visit previously unseen states, which could cause catastrophic failures. Additionally, the subsequent labeling by the human

can be suboptimal both in the amount of data recorded (perhaps recording more data in suboptimal states than is needed to learn an optimal policy) as well as in capturing the intended result of the human observer's action (as in distinguishing a minor course correction from a sharp turn, or the appropriate combination of actions to perform a behavior). Another limitation of DAgger is that the human feedback was provided *offline* after each run while viewing a slower replay of the video stream to improve the resulting label quality. This prevents the application to tasks where real-time interaction between humans and agents are required.

Demonstrations can also be used to infer the cost function used by the demonstrator while performing the task, known as Inverse Optimal Control (IOC) [28, 29, 30, 31] or Inverse Reinforcement Learning (IRL) [32, 33, 28] when it is inferred the reward function, which is the negative of the cost function. Following this principle, related research uses the maximum entropy principle to maximize the entropy of the model distribution subject to the feature constraints from demonstration data [34, 35], evaluated using the difference between value function for the optimal policy obtained using the learned reward model and using the ground truth reward, or expected value difference [35]. Leveraging the principle of maximum entropy, previous approaches [36] learns the cost function and optimizes a policy for it at the same time, compares this policy to the demonstrated trajectories, performs and evaluation step, and repeats the procedure until convergence or the desired performance is achieved. Alternatively to IRL, demonstrations can be integrated directly to Reinforcement Learning (RL) algorithms to increase their sample-efficiency during training.

1.2.1.2 Learning from Interventions

Learning from interventions (LfI), where a human acts as an overseer while an agent is performing a task and periodically takes over control or intervenes when necessary [37, 38], can provide a method to improve the agent policy while preventing or mitigating catastrophic behaviors [39]. Similar work by Hilleli and El-Yaniv [40] has proposed using human interaction to train a classifier to detect unsafe states, which would then trigger the intervention by a safe policy previously trained based on human demonstration of the task. This technique can also reduce the number of direct interactions with the agent, when compared to learning from demonstration. However, this technique

suffers from the disadvantage that desired behaviors must be discovered through more variable exploration, resulting in slower convergence and less stable behavior.

Related work includes slowing down the execution of the task controlled by RL agent during training so the human can intervene at any step to prevent catastrophic actions by replacing agent's actions by safe human actions. On Saunders et al. [39], the task is paused and a model is trained to imitate human intervention decisions. The trained intervention model replaces human and the training continues. They show that this approach works well for simple cases when the human is able to prevent catastrophic actions but it does not scale well to more complex tasks due to the amount of human intervention required. Related work uses interventions to build upon policies trained with human demonstration for robot navigation in the real world [41] and kinesthetic teaching of a robot via keyframes that can be connected to generate trajectories [42].

Several cases include the combination of demonstrations and mixed initiative control for training robotic polices [43] as well as combining imitation learning with interactive reward shaping in a simulated racing game [40]. Previous approaches combined human actions with robot autonomy to achieve a common goal. On Javdani et al. [44], the robot did not know the goal a-priori and used inverse optimal control and the maximum entropy principle to estimate the distribution over the human's goal based on previous inputs. They showed this approach enabled faster task completion using less data samples when compared to the traditional approach where the robot first predicts the goal then assist for it. Other approaches [45] combined a pre-trained Q-learning policy with human input and showed that the combination is better than the parts by themselves. The main limitation is that it needs a Q-function trained already, which might not be realistic for real-world tasks.

1.2.1.3 Learning from Evaluations

Learning from evaluation (LfE) is one such way to leverage human domain knowledge and intent to shape agent behavior through sparse interactions in the form of evaluative feedback, possibly allowing for the approximation of a reward function [46, 47, 48]. This technique has the advantage of minimally tasking the human evaluator and can be used when training behaviors they themselves cannot perform, only requiring an understanding of the task goal. An example would be

maneuvering a robotic arm with multiple degrees of freedom in a constrained space. Due to the number of joints and obstacles, the human might not be able to provide complete demonstrations but is able to evaluate if the maneuver was executed successfully or not. Additionally, if the time-scale of the autonomous system is faster than human reaction time, then it can be challenging for the autonomous system to attribute which actions correspond to the provided feedback (credit assignment problem).

Similar to LfI, it can be slow to converge as the agent can only identify desired or even stable behaviors through more random exploration or indirect guidance from human negative reinforcement of unwanted actions, rather than through more explicit examples of desired behaviors. Another disadvantage is that the human evaluation signals are generally non-stationary and policy-dependent, for example, what was classified as a good action in the past may not be classified in the same way in the present depending on the human perception of the autonomous system's policy.

Thomaz and Breazeal [49, 50] addressed how humans want to reward machines guided by RL algorithms, how to design machines that learn effectively from natural human interaction, and the motivation to maintain safe exploration of new actions and states when using RL algorithms guided by humans. Knox et al. [46, 52, 51, 53] have developed methods for adapting human inputs to classic machine learning algorithms via the "Training an Agent Manually via Evaluative Reinforcement" (TAMER) framework. The human trainer rewards the robot based on its past actions and the framework handles the distribution of these rewards along the state-action pairs to shape the policy of the intelligent agent. This work was later extended to use deep neural networks as learning representation to solve ATARI games using raw images as inputs, called Deep TAMER [48]. León et al. [54] mixed human demonstration and direct natural language human feedback during the task execution as an additional dynamic reward shaping mechanism. So far these methods have only been applied to low-dimensional RL problems.

Differently than using the human as a reward function generator, MacGlashan et al. presents the Convergent Actor-Critic by Humans (COACH) algorithm, which uses the human as the temporal difference (TD) error. COACH is based on the insight that the advantage function is a good model

of human feedback and that actor-critic algorithms update a policy using the critic's TD error, which is an unbiased estimate of the advantage function. This work was later extended by Arumugam et al. [55] to use deep neural networks, called Deep COACH, to learn policies mapping raw pixel inputs to actions in Minecraft.

Christiano et al. [56] provided a framework to shape the behavior of an RL agent when the task is relatively complicated to be demonstrated by a human operator. Instead of performing the task, the agent presents two different trajectories (sequence of states and actions) and queries the human the preferred option. The resultant behavior is more natural when compared to the one achieved by human-handcrafted rewards. Similar to this approach, Singh et al. [57] presents VICE-RAQ (Variational Inverse Control with Events - Reinforcement learning with Active Queries) which constructs a classifier to function as a reward function based on example of successful outcomes provide by humans instead of relying on a handcrafted environment reward. After that, the agent can query the human for more labels when it encounters possible goal states. Ibarz et al. [58] combines learning from expert demonstrations to learning from preferences to solve ATARI games without using the game score. They train an behavior cloning policy then learn a reward using human feedback and trajectory preferences and show that this combined approach outperforms the use of only preferences or only demonstrations. Additionally, they also show that humans can prevent reward hacking due to the constant and online feedback.

1.2.2 Reinforcement Learning

This section addresses the state-of-the-art and current challenges relevant to the Reinforcement Learning (RL) literature, including sample-efficiency when training RL algorithms and deploying these algorithms in real-world and robotics applications.

1.2.2.1 State-of-the-Art Algorithms

Reinforcement Learning is a fast-moving field and according to Schulman [59], there is still no consensus on which RL algorithm would perform better on different applications and robotic systems: actor-only and actor-critic methods, policy gradient methods (e.g., score function, natural

or non-natural re-parameterization), value learning (e.g., Q-learning and its derived algorithms), or derivative-free optimization approaches (e.g., cross-entropy method). According to Grondman et al. [60], in (quasi-)stationary Markov Decision Processes (MDP), actor-critic methods should provide policy gradients with lower variance than actor-only methods. However, actor-only methods are more resilient to fast changing non-stationary environments due to the critic not being able to adapt as quick as the actor to the new environment and, consequently, providing poor information for actor updates.

It could be argued that policy gradient reinforcement learning started with Sutton et al. [61], work that paved the way to modern policy gradient algorithms using function approximation. Another classic work by Kakade and Langford [62] proposed an algorithm to estimate the lower bound of a policy performance, which can be used to constrain the policy updates and guarantee policy improvement. Policy gradient is a sub-field of policy search [63] where the gradient is used to guide the search. Alternatively, Levine and Koltun [64] uses iterative linear quadratic regular (iLQR [65]), initialized from demonstrations, for trajectory optimization and direct policy search. The iLQR samples trajectories from high-reward regions, which are incorporated to the policy search by using regularized importance sampling.

In modern policy gradient methods for deep reinforcement learning (Deep RL), Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG) by Schulman et al. [66] iteratively optimize policies with guaranteed monotonic improvement using the natural policy gradient to optimize a given model in the policy-space, constrained by a policy trust region, instead of optimize it in the parameter-space performed by gradient descent. Similarly, Proximal Policy Optimization (PPO) [67] incorporates similar ideas to TRPO in terms of constraining policy updates in the policy-space instead of parameter-space by clipping the objective function and also penalizes the KL divergence between new and old policy in the objective function. The bias-variance trade-off for this methods can be controlled using Generalized Advantage Estimator (GAE) [68], leading to high-performance agents in complex controls benchmarks. Multiple Deep RL methods for continuous control are benchmarked by Duan et al. [69] showing the efficacy of the natural policy

gradient methods.

Policy gradient methods can also be parallelized. Heess et al. [70] implements a distributed form of PPO where data collection and gradient calculation are distributed between parallel workers. The authors show that agents trained in rich environments without an specifically handcrafted reward function can lead to the development of non-trivial locomotion skills that would be difficult to be encoded in a reward function. Mnih et al. [71] introduces the Asynchronous Advantage Actor-Critic (A3C) that substitutes the experience replay buffer by running multiple copies of the environment. A3C maintains a policy and a value function and uses a mix of n-step returns to update both. Policy is updated using the policy gradient based on advantage function, while the value function is updated based on the mean squared error with the n-step return. This work also uses target networks to stabilize training, shared layers between policy and value function, and an entropy regularization term with respect to the policy parameters. The IMPALA (Importance Weighted Actor-Learner Architecture) algorithm by Espeholt et al. [72] generates, in parallel, multiple trajectories and communicate them to a central learner. Since the policy generating the trajectories might be different to the one being updated, they propose the new V-trace algorithm for off-policy correction using truncated importance sampling. This new approach achieves better performance than previous agents with less data on controls and ATARI benchmarks.

For value learning in discrete action-spaces, for example, when the agent learns to predict based on the temporal differences [73] or learns a Q-function [74] representing the expected discounted sum of rewards to be received at the end of an episode when the agent is a given state and performs a given action, much of the recent progress came after the work by Mnih et al. [75, 76], which successfully integrated deep neural networks to reinforcement learning in the Deep Q-Network (DQN) algorithm to solve multiple ATARI games learning directly from images (pixel inputs) — a significantly larger state-space when compared to learning from low-dimensional states. Key insight from the DQN work is the benefit of having a replay buffer [77] to storage previous interactions between agent and environment, more specifically using Prioritized Experience Replay (PER) [78]. By using an experience replay and random sampling, the samples become closer to satisfy

the i.i.d. (independent and identically distributed) assumptions used in stochastic gradient-based algorithms and liberates online learning agents from processing transitions in the exact order they are experienced [78]. In addition to that and to replace random sampling, PER samples transition with higher TD-error, corrected with importance sampling, and diversity alleviated using stochastic prioritization, liberating agents from considering transitions with the same frequency that they are experienced [78]. Successful transitions can also be added to this buffer to aid the learning process [79].

The DQN work was extended by van Hasselt et al. [81, 80] with the Double Deep Q-Networks (DDQN) algorithm, correcting the overestimation of the Q values typical of DQN approach due to the maximization step over estimated action values. DDQN adds a second Q network (called target network) that decouples action selection and action evaluation and leading to better performance on the ATARI benchmark. Further extending DQN, Wang et al. [82] proposes a neural network architecture that decouples the Q-function in values and advantages while sharing a common feature learning module. This switches the focus to finding the more valuable states and removing the need to evaluate the Q-function for each action, improving DQN performance on tasks with larger action-space. Instead of only learning the expected value for the value functions, Bellemare et al. [83] with Distributional Q-learning proposes learning complete value function distributions. Authors argue that learning the complete distribution, when combined with function approximation, leads to a more stable policy during training. Many of these advances and extensions to DQN were combined in a single algorithm, Rainbow DQN [84], to achieve state-of-the-art performance on ATARI games. Rainbow DQN uses noisy networks for exploration [85], instead of the traditional ϵ-greedy approach used in Q-learning which adds noise to the action-space. In the work by Fortunato et al., the authors add noise to the parameters of the policy (to the weights, if using a neural networks), which are also learned with gradient descent. This approach lead to better performance when compared to traditional RL approaches for exploration.

For value learning in continuous action-spaces, the Deep Deterministic Policy Gradient (DDPG) [86, 87] by Lillicrap et al. was one the first adaptations of DQN for continuous action-spaces. DDPG

uses two deep neural networks to learn at the same time a deterministic policy and an Q-function, which is assumed to be differentiable with respect to the actions and used to guide the policy updates based on its gradients, substituting the $\max_a Q(s, a)$ used in DQN. Alternatively to DDPG, Gu et al. [88] proposes the Normalized Advantage Function (NAF) algorithm that parameterizes the advantage term as a quadratic function so the actions are computed analytically and the agent only needs to learn the Q-values. The authors also show that incorporating a model-based approach to their method to generate off-policy samples during the beginning of training leads to better performance when compared to just using a model-free approach. Interestingly, the model-based part needs to be switched off a later stages of training because Q-learning performs better with on-policy samples when the policy is already better developed. Similar to DQN, the main problem with DDPG is overestimating the Q-function and having the policy to exploit this error, leading to policy breaking during training. Fujimoto et al. [89] presents the Twin Delayed DDPG (TD3) algorithm that solves this issue by using two clipped Q-networks, learning the policy slower than the critic, and adding noise and smoothing actions to prevent exploitation of overestimated Q-values. This approach leads to better performance when compared to DDPG. As an alternative to the "deterministic policy" of DDPG, Haarnoja et al. [90, 91] introduces the Soft Actor-Critic (SAC) algorithm combining off-policy actor-critic training with a stochastic actor, and further aims to maximize the entropy of this actor with an entropy maximization objective. The entropy term on the objective functions allow to control for the exploration-exploitation trade-off in RL while learning a more robust policy to noisy observations.

Hybrid approaches as the Interpolated Policy Gradient (IPG) algorithm [92] and Q-Prop [93] integrate the sample-efficiency of off-policy RL algorithms with the stability of the on-policy ones by combining off- and on-policy updates in the same loss function while satisfying performance bounds. These algorithm unifies both techniques, has theoretical guarantees on the bias introduced by off-policy updates, and improves on the state-of-the-art model-free deep RL methods.

1.2.2.2 Sample-Efficiency in Reinforcement Learning

Current state-of-the-art RL algorithms heavily rely on multiple graphics and central processing units (GPUs and CPUs) to train intelligent agents on end-to-end approaches [75, 80, 78, 71]. End-to-end algorithms are initialized with no previous knowledge of the task nor the environment and and the action selection process (trial-and-error) develops almost randomly. This approach lead to learning algorithms that are able to generalize to multiple problems at the cost of high number of interactions with the environment. Agents learning simple continuous tasks (e.g., controlling a simulated two-link robotic arm) require on average more than 2.5 million samples to achieve satisfactory performance [87]. On more complex tasks (e.g., control of a simulated humanoid robot or robots with multiple degrees of freedom) they require on average 25 million samples [69]. Discrete computer game tasks, such as learning how to play Atari games, may require tens of millions of interactions with the environment to achieve state-of-the-art results [75, 94]. Less complex algorithms, like REINFORCE [95] and Cross-entropy Methods (CEM) [96], achieve good performance optimizing policies parametrized as deep neural networks, but are likely to converge prematurely to local optima [97]. There is a need for sample-efficient learning architectures in which a learning agent would quickly learn meaningful behavior, requiring fewer interactions with the environment, easing the transition to robotic systems performing real-world tasks.

Model-based approaches are a common option to improve sample-efficiency of RL algorithms but it becomes challenging to learn a complex dynamical system with a low number of samples available. Feinberg et al. [98] uses model-based approaches to estimate value functions for short term and model-free for long-term estimation of Q-values in reward dense environments. The distinction between how many time steps comprises short- or long-term relies on model value expansion (MVE) for value estimation error. Their approach can be generalized for any actor-critic algorithm. Nagabandi et al. [99] proposes to learn the dynamics using state and action as input and the variation in the state as output of a deep neural networks based on data generated by a random policy. With this learned model, at every time step the agent plans the trajectory using model predictive control (MPC) [100, 101], performs one step of the trajectory and re-plans. Reward is

based on how close the agent follows the trajectory commanded. This on-policy data is aggregated with the initial data used to learn the model and the dynamics is retrained once in a while. On their experiments with a real-robot, the model-based approach performs worse than a model-free benchmark. To solve this issues, the authors fine-tune the MPC controller together with learned dynamics using model-free RL. First, the agent's policy is trained to replicate the MPC controller (still mixes on- and off-policy data to re-train policy and dynamic model). At the final stage this policy is transferred to a actor-only algorithm, for example TRPO, and the training continues. This approach outperforms the model-based or model-free alone.

Another approach for leveraging simulated rollouts is proposed by Ha and Schmidhuber [102]. On this work, the agent learns on a unsupervised manner, from random rollouts, a compressed representation of the environment using variational autoencoders (VAE) [103, 104], mixture density networks (MDN) [105, 2], and recurrent neural networks (RNN) [106, 107, 108], and use this learned model to update its policy which is later performs in the real environment. This research is inspired in human factors research that suggests that human brains only remembers certain features and what humans perceive depends on brain's internal model for future prediction. The models learned are mostly imperfect but the authors uses a weighting parameter to quantify the amount of uncertainty. The policy is trained using Covariance-Matrix Adaptation Evolution Strategy (CMA-ES) [109, 110] using multiple CPU cores to run multiple copies of the environment to parallelize and accelerate learning. Another model-based approach for imperfect models by Racanière et al. [111] called Imagination-Augmented Agents (I2A) does not rely exclusively on simulated returns. Their approach still learns a forward model but converts trajectories to embeddings, abstracting model imperfections.

Dynamic models can also be use to control exploration of RL environments. Pathak et al. [112] introduces an intrinsic reward module that rewards the agent based on how well it is able to predict the environment states. The better the prediction, lower the reward. This forces the agent to explore unexplored areas in the state space. In their approach, the agent creates a forward model based on latent-space and also learns an inverse model to predict which actions were used to change states.

Similar model-based approach was earlier proposed by Schmidhuber [113] where the agent builds the model by actively provoking situation where it expects to learn something about the dynamics of the environment.

Other approaches to increase sample-efficiency of RL algorithms include combining them with classical controllers [114], adding auxiliary task and rewards [115], and learning from episodes where the goal is not reached [116]. In the work by Johannink et al. [114] the authors add actions produced by the RL policy to actions of a hand-tuned classic controller. The majority of the task is solved by the classic controller and the rest is fine-tuned with RL. In their paper they use the TD3 algorithm to solve a block placing task with handcrafted dense reward function using simulated and real robots, also transferring policies learned in simulation to real world. The state-space for the RL agent includes the goal of the task and action-space is the position of the end-effector. The addition of auxiliary tasks in the work by Jaderberg et al. [115] introduces the UNsupervised REinforcement and Auxiliary Learning (UNREAL) agent, which translates to adding two additional reward signals (in addition to the one returned by the environment): maximize change in pixel intensity (generally correlated to important events in the screen) and correct reward prediction based on stacked input frames. It does increase the sample-efficiency because the agent is able to extract more information from the same amount of data plus converts any tasks with sparse reward signals to dense, aiding the optimization process. The main insight to learn from episodes where the goal is not reached comes from Andrychowicz et al. [116] where the Hindsight Experience Replay (HER) algorithm is proposed. HER adds a goal vector to standard observation input for RL algorithms. At the end of each episode, it interprets the final state as a pseudo-goal so even when the agent fails to achieve the desired goal, it at least learns how to achieve this alternative state. It solves to problem of sparse rewards tasks with narrow successful regions where positive rewards are difficult to achieve.

1.2.2.3 Human-in-the-loop Reinforcement Learning

Reinforcement learning has been proven to work on scenarios with well-designed reward functions and easily available interactions with the environment [8]. However, in real-world robotic applications, explicit reward functions are non-existent, and interactions with the hardware are

expensive and susceptible to catastrophic failures [8]. This motivates leveraging human interaction to supply this reward function and task knowledge, to reduce the amount of high-risk interactions with the environment, and to safely shape the behavior of robotic agents [8].

A common approach in human-in-the-loop reinforcement learning is modify the reinforcement learning loss function to leverage a human dataset of trajectories to solve the desired task. Hester et al. [117] presents the iconic Deep Q-learning from Demonstrations (DQfD) algorithm where the agent is pre-trained and trained with four combined losses: 1-step double Q-learning loss, n-step double Q-learning loss, supervised large margin classification loss, and L2 regularization on network weights and biases. Authors argue that the combination of all four losses during pre-training is essential to learn a unified representation that is not destroyed when the loss function changes from the pre-training to the training phase. Even after pre-training, the agent must continue using the expert data [117]. They also modified the replay buffer to combine expert demonstrations and self-generated data (never overwriting the expert data) sampling proportional amounts of each type. The authors shows that DQfD lead to more sample-efficient approach to solve the ATARI benchmarks.

Similar to DQfD but without the pre-training phase and the supervised loss in the loss function Večerík et al. [118] present the Deep Deterministic Policy Gradient from Demonstration (DDPGfD) algorithm. It loads all expert demonstrations in a prioritized experience replay buffer before training, which is kept throughout the training process, and performs more than one learning step when sampling data from the buffers. The main contribution of this algorithm is that it can be used in continuous action-spaces, while DQfD is restricted to discrete ones. Another extension to DQfD proposed by Pohlen et al. [119], the Ape-X DQfD algorithm differs from DQfD in three aspects: no pre-training phase using only expert transitions, fixed ratio of actor and expert transitions, and the supervised loss is only applied to the best expert episode instead of all episodes. This was the first deep RL algorithm to solve the first level of Montezuma's Revenge, the ATARI game in which DQN had poor performance [76]. A pre-training strategy is also used by Cruz Jr et al. [120] feature and policy learning in RL. The authors tackle the feature learning during the pre-training

phase with a combined supervised classification loss, an unsupervised reconstruction loss, and a value function loss. They also use a transformed Bellman operator to scale the whole action-value function without clipping or modifying the reward signal. Another contribution is their self-imitation learning approach encourages the agent to imitate past decisions only when the returns are larger than the current estimated value. Their approach is tested on ATARI games and shows improved sample-efficiency when compared to deep RL algorithms.

Building on top of Deep Deterministic Policy Gradients and Hindsight Experience Replay, Nair et al. [121] shows that when an actor is pre-trained with demonstration off-policy data and transferred to RL for training, the actor weights are destroyed by the untrained critic. Also, training the critic with only off-policy data it fails to correctly evaluate on-policy transitions, which motivates the combined loss approaches and mxiture between on- and off-policy data. Similarly, the approach by Sun et al. [122] combines expert and agent data in a mix of on- and off-policy updates to reduce distribution mismatch between training and testing dataset and argue that interactive approaches leads to better performance through a reduction to no-regret online learning. This algorithm matches expert performance and even surpass it in case the expert is not optimal. Multiple critics can also be used to constrain the actor's policy, as showed by Durugkar et al. [123] with the Multi-Preference Actor Critic (M-PAC) framework.

Mixing directly expert trajectories with RL, Schoettler et al. [124] extends the concept of Residual RL, where actions selected by the agent are added to actions performed by an expert controller, to the setting of vision-based manipulation tasks. The authors show that this approach can be used to train agents directly in the real-world. Peng et al. [125] uses expert trajectories as reference to compute a reward function that penalizes mismatch between reference and performed trajectories. Constructed this reward function, the agent can be trained with any RL algorithm. The authors show that multiple references can be integrate into the learning process to develop multi-skilled agents capable of performing multiple and combined skills.

Humans can also aid RL algorithms through natural language. Kaplan et al. [126] trains an agent that learns the meaning of English words, translates them to game states, and learns how

to execute these commands. Game images are used to train a policy and state value function using an state vector augmented with language instructions. The authors generate training data by template matching the frames with the instructions, which is later used to verify if instructions were completed, giving the agent an additional reward. They also generated training data by playing the game and leaving the agent at specific positions or performing specific actions to cover gaps in the demonstration dataset. Similarly, Waytowich et al. [127] proposed a model that learns a mutual embedding between natural language commands and goal-states that can be used as a form of reward shaping and improve RL performance on environment with sparse rewards. Their approach is tested on the SC2LE (StarCraft II Learning Environment) [128]. For dialogue applications, Jaques et al. [129] train a generative model on known sequences of interaction data that is later used as reference during the RL training phase, penalizing divergence from this prior generative model with different forms of KL-control. Authors argue that this is necessary to remove the overestimation bias when agents are pre-trained directly with expert data.

Humans be queried to provide feedback, called active learning, and train a reward model for RL agents. Daniel et al. [130] propose to use humans to evaluate the agent's actions (assign numerical values to observed trajectories), instead of having them to design the reward function. Since human evaluation is noisy and non-repeatable, the authors also propose to learn a probabilistic model of the reward function using Gaussian processes and a noise hyperparameter. The reward model is learned by sampling from a memory buffer the best performing trajectories plus the trajectories evaluated by the human expert. When to query the expert is based on the ratio of the predictive variance and observation noise, called lambda. If lambda is above a certain threshold, the agent queries the human and uses this evaluation to update the reward model. The proposed approach is evaluated in a simulated environment and in a real robot for grasping tasks. Su et al. [131] use RL to train a dialogue policy at the same time as a reward model is trained based on human feedback. They also use Gaussian processes to quantify uncertainty to reduce number of queries to the user. This allow for online learning in real-world dialogue systems without manually annotated data. Lopes et al. [132] introduces active reward learning for inverse reinforcement learning where the agent queries

the user for demostrations at specific areas of the state-space. It does that by measuring a state-wise entropy in the distribution and queries the user for the most informative states. Including elements of active learning, Hilleli and El-Yaniv [133] initially use human to generate expert trajectories and train a policy to imitate them. The human later labels the states in order to train a reward model and uses DDQN deep RL algorithm to optimize the reward, which learns to perform better than the human demonstrator.

More recent meta-learning approaches, as proposed by Zhou et al. [134], learns initially from few demonstration and later from binary feedback. This approach works with two separate policies: trial (learn from demonstrations) and retrial (learn from experience) policy. Trial policy is trained in a meta-imitation learning setup and is frozen to generate data to the retrial policy, which is trained on this stationary data. Huang et al. [135] address the one-shot imitation learning problem, where the goal is to execute a previously unseen task based on only one demonstration. This is made possible by formulating the one-shot imitation learning as a symbolic planning and symbol grounding problem together with relaxation of the discrete symbolic planner to plan on the probabilistic outputs of the symbol grounding model.

1.2.2.4 Reinforcement Learning in Robotics and in the Real World

In robotics, it is often challenging to successfully integrate learning algorithms and controllers. Valasek et al., Valasek et al. [136, 137], developed a novel architecture called Adaptive-Reinforcement Learning Control (A-RLC) for morphing air vehicles. It defines optimal aircraft shape based on mission objectives, learns how to morph to the desired format, and controls variations on system's dynamics due to the new shape.

Reinforcement learning methods often fail in dynamical systems due to failure in properly representing the problem, inaccurate function approximation, and not accounting for time dependency in the selection of actions, as pointed by Kirkpatrick and Valasek [138]. Controlling real continuous systems requires computer-based control, so sampling of the continuous system is necessary [138]. If necessary to discretize these continuous systems, the size of the grid must be tuned to capture environment details and still achieve good convergence, as presented by Lampton et al. [139].

Often applied to simulated tasks and environments, RL algorithms need to overcome their extensive trial-and-error approach before they can be applied to real robotic systems. Kober et al. [140], in their extensive survey of reinforcement learning applied to robotics, highlighted several open questions that need to be answered in order to have RL techniques reliably deployed to robotic systems, for example: 1) How to choose the best system representation in terms of approximate states, value functions, and policies, when dealing with continuous and multi-dimensional robotics problems? 2) How to generate reward functions straight from data, reducing the need for manually-engineered solutions? 3) How to incorporate prior knowledge and how much is needed to make robotics reinforcement learning problems tractable? 4) How to deal with parameter sensitivity and reduce the need for preprocessing sensory data, simplifying behavior programming? 5) How to approach model error and under-modeling during simulated training — when the simulated environment cannot perfectly model every aspect of the real environment, since the policies learned only in simulation frequently cannot be transferred directly to the robot? 6) How to develop accurate simulation training environments which will ease transfer of policies learned from simulation to real-world hardware tasks?

Amodei et al. [141] addressed concrete risks of poorly-design AI systems deployed to real-world applications. Specifically in RL, *reward hacking* and *safe exploration* are the main issues. Reward hacking refers to developing unintended behavior by exploiting a faulty reward function. For example, a running robot that is rewarded according to its speed could learn to run in circles at maximum speed instead of completing the running course. Safe exploration refers to developing unsafe behavior (e.g., behavior that would damage the agent or its environment) while exploring the observation space. These problems are currently unsolved.

There is also a research gap on *trust on autonomous systems*. Prior studies [142] indicate that humans have low levels of trust in semi and fully autonomous systems. This is directly correlated with the unpredictability of the robot [143], their rate of success and reliability [144, 145, 146], and how knowledgeable the user is about an autonomous system [147] and the task to be completed [148]. This highlights the need to develop a platform to illustrate what future actions are being

planned by the intelligent agent or any other form of education to demystify the intelligent behavior. This would improve human trust in these systems and allow humans to supervise future actions and consequences.

1.3 Motivation and Objective

1.3.1 Motivation

As discussed on Sections 1.2.1 and 1.2.2, current state-of-the-art research on learning algorithms focuses on end-to-end approaches. The learning agent is initialized with no previous knowledge of the task nor the environment and the action selection process (trial-and-error) develops almost randomly. End-to-end approaches abstract intermediate learning milestones at the cost of high number of interactions with the environment. An important question is how RL agents could incorporate prior knowledge or learn directly from a trained policy. For example, incorporate human knowledge or learn from an existing sub-optimal controller. Humans are preferred because they dictate the task to be accomplished and have traits that are in the frontier of learning algorithms research, as for example long-term autonomy [149] and continual lifelong learning [150].

Currently, there is no definite approach to combining humans and RL agents. Machines that can be taught by and seamlessly work in collaboration with humans will not only learn faster, but will potentially expand the frontier of having machines augmenting human performance. The rapid increase of automation and development of AI techniques demands a better understanding of how humans and machines can better work in collaboration.

1.3.2 Objective

The objective of this research work is to develop the theoretical foundation to integrate multiple human input modalities (in the form of demonstration, intervention, and evaluation) to allow intelligent data-driven agents to learn in real-time, safely, and with fewer samples (increase sample-efficiency) by bootstrapping human interaction data and reinforcement learning theory.

1.4 Research Contributions

This dissertation investigates how to efficiently and safely train autonomous systems in real-time by extending supervised and reinforcement learning theories and human input modalities. Intuitively named the **Cycle-of-Learning** (CoL), this research investigates and develops training autonomous system policies through human interaction that is based on the evidence-based teaching methods of “teaching for mastery" [151]. On teaching for mastery, or mastery learning [151, 152, 153], human interaction is provided systematically, from complete human control (and no autonomy) when learning from demonstrations to no human control (and full autonomy) at the reinforcement learning stage. After learning from demonstration the policy is evaluated in real-time and given feedback only on specific areas that needs to improve through interventions. After mastering an initial concept, the role of the human is only to evaluate the performance of the policy while it develops by itself using reinforcement learning. This process is repeated as new concepts are introduced and the policy masters previous concepts, as illustrated in Figure 1.1.

While extensive research has been conducted into each of these stages separately in the context of machine learning and robotics, a complete theoretical integration of these concepts has yet to be done. This theoretical integration will be important to fielding adaptable autonomous systems that can be trained in real-time to perform new behaviors depending on the task at hand, in a manner that does not require expert programming.

The contributions of the research are:

1. A novel theoretical contribution of incorporating multiple human input modalities to reinforcement learning algorithms to perform, simultaneously, on-policy and off-policy learning, leveraging their strengths and mitigating their weaknesses.

2. Reduction of the data-complexity and the number of samples (interactions with the environment) required by state-of-the-art reinforcement learning algorithms to achieve satisfactory performance.

3. Enabling reinforcement learning agents to be trained in real-time, with real-world data, safely.

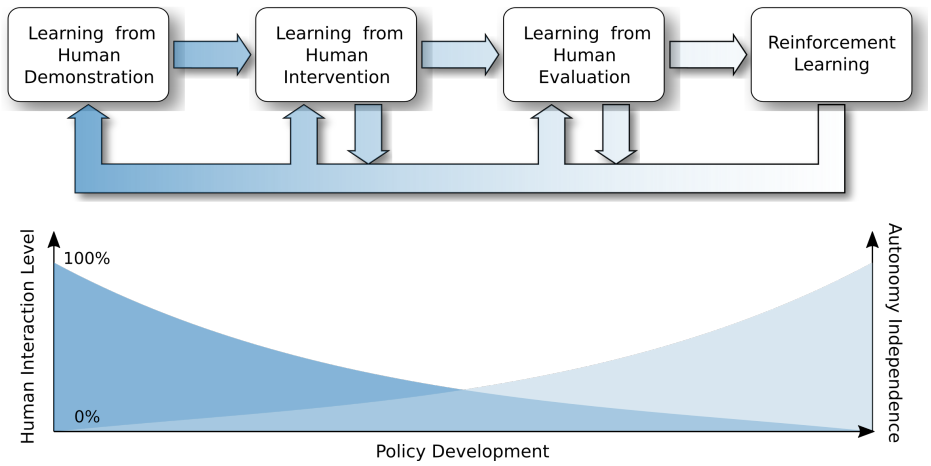


Figure 1.1: Cycle-of-Learning for Autonomous Systems from Human Interaction: as the policy develops, the autonomy independence increases while the human interaction level decreases. Adapted from [1].

Results presented in this dissertation show that policies trained with human demonstrations and human interventions together outperform policies trained with just human demonstrations while simultaneously using fewer data samples. To the best of knowledge, this is the first result showing that training a policy with a specific sequence of human interactions (demonstrations, then interventions) outperforms training a policy with just human demonstrations (controlling for the total amount of time of human interactions). One can obtain this performance with significantly reduced data requirements, providing initial evidence that the role of the human should adapt during the training of safe autonomous systems.

1.5 Organization

This dissertation is organized as follows. Chapter 2 details Behavior Cloning and Imitation Learning in discrete or continuous spaces and how these algorithms can be used to replicate human behavior. Chapter 3 formalizes the Reinforcement Learning problem as a Markov Decision Process and explains how this research field transitioned from shallow to deep representations for policy and value functions when using function approximation. This chapter also includes an unusual application example: Deep Reinforcement Learning on Intelligent Motion Video Guidance for

Unmanned Air System Ground Target Tracking. Chapters 4 and 5 presents to case studies of Deep Reinforcement Learning applied to Aerospace problems: the tracking of a ground target using unmanned aerial systems and learning to control morphing wings. Chapter 6 initializes the human-in-the-loop work with a case study leveraging Inverse Reinforcement Learning methods to land a simulated lander and simulated unmanned aerial vehicle by learning from human demonstrations of the task. Chapter 7 addresses how to leverage initial human demonstrations to learn the intrinsic reward function used by the human to pursue the task goal while performing the demonstrations. This reward signal is then used to feed reinforcement learning algorithms in a unmanned aerial system collision-avoidance scenario. Chapter 8 introduces the main theoretical contribution of this dissertation: Cycle-of-Learning for Autonomous Systems from Human Interaction. It covers how to combine multiple forms of human interaction and integrate them to reinforcement learning. Chapter 9 explains how to efficiently combine human demonstrations and interventions on learning algorithms to increase task performance while using fewer human samples. The main study case is learning from human interaction autonomous landing controllers for unmanned aerial systems. Chapter 10 finalizes the Cycle-of-Learning by transitioning from policies learned from human interaction to reinforcement learning. This approach is validated on the previous unmanned aerial system landing scenario and standard reinforcement learning tasks for continuous control. Conclusions from the various applications and recommendations for future work are summarized in Chapter 11.

2. MACHINE LEARNING AND CLONING BEHAVIORS

This chapter introduces the notation and basics of machine learning, specifically supervised learning, and how an algorithms can learn to clone behaviors by framing the learning problem as a sequential decision making problem. This includes important concepts in machine learning, the mathematical formulation of sequential decision making problem framed as a Markov Decision Process, the relation between imitation learning and behavior cloning, success cases in the literature, and the limitations of this approach in isolation.

2.1 Problem Definition

According to Barber [154], Machine Learning is the research field concerned in automating large-scale data analysis, leveraging concepts of statistics with focus on mathematical modeling and prediction. The dataset that one uses to learn from to create this mathematical model, or *train the model*, is called the *training set* [2]. Ideally, the model trained on this training set will be able to *generalize* to new unseen inputs, often called the *test set* [2]. The process of training the model depends on how this training set is structured, which further subdivides the field of machine learning in three groups: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. In supervised learning, the training set contains data inputs and desired outputs so the learning model changes its internal parameters during training in order to produce outputs closer to the desired ones given the same input data. For example, a machine learning model for autonomous vehicles would be interested in identifying traffic signs and lights from image data so the vehicle can behave according to the traffic rules. The training set would consist in multiple images as input and the location of all traffic signs and lights in the image as output. In unsupervised learning, the training set consists only of input data and no desired output. The main goal is to discover groups in similar category (clustering), estimate the input distribution (density estimation), project data in different dimensions to aid visualization [2]. An example of unsupervised learning would be a machine learning model that is trained to cluster customers of a bank based on their credit card

transaction history. In Reinforcement Learning there is no training set and machine learning model is trained by trial-and-error while performing a desired task in order to maximize a performance metric. A example of reinforcement learning would be a machine learning model that receives images from a screen while playing a computer game and aims to maximize the game score by performing different commands and observing its effect in the next game screens and score.

Supervised learning is covered with more details on Section 2.4 and Reinforcement Learning on Chapter 3 of this research work. Unsupervised learning is out of scope of this dissertation however introductory material in the area can be found in Barber [154] and Bishop [2].

2.2 Supervised Learning

Barber [154] defines supervised learning problem as, given a set of data pairs

$$\mathcal{D} = \{(x^n, y^n), n = 1, \ldots, N\},$$

learning the relationship between the input x and output y such that, given an unseen input x' not present in $\mathcal{D}$ the predicted y' output is accurate, assuming the pair (x', y') is generated by the same unknown process that generated the set $\mathcal{D}$. In other words, it is desired to predict y' conditioned on know x' and the set $\mathcal{D}$, which can be represented by the conditional distribution $p(y'|x, \mathcal{D})$. If the output y is one of a discrete number representing a possible discrete outcome, the supervised learning problem is called "classification" (for example, given computed tomography (CT) x-ray images of the lung, predicts if the person is likely or not to have cancer [155]). If the output y is a continuous variable, the supervised learning problem is called "regression" (for example, predicting future market price of a given stock based on financial news articles [156]). This distinction is important to define the loss function used to compute the misfit between true and predicted model, which guides the optimization process used to learn the model, to be discussed in more detail throughout the next sections.

2.2.1 Model Representation and Learning

The word *model* has not been properly defined in this work yet. The model is the mathematical representation of the underlying function or distribution it is desired to approximate, from which the training set is sample from, in order to generalize to unseen data samples and predict future outcomes. Models differ by the number and the mathematical relationship between its parameters, which should be chosen based on the underlying function the model is desired to approximate and assumptions made. Examples of these assumptions are if the underlying process that generated the training set is linear or nonlinear, noise processes that affect the measurements, and others.

As explained by Bishop [2], an example of model representation would be a polynomial function of the form

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j, \tag{2.1}$$

where M is the order of the polynomial and its coefficients $w_0, \ldots, w_M$ are collectively denoted by the vector $\boldsymbol{w}$. Even though there is a linear relation between the polynomial coefficients $\boldsymbol{w}$, the model $y(x, \boldsymbol{w})$ is nonlinear and, consequently, is able to approximate nonlinear functions. The main limitation of models comprised of linear combination of fixed basis is that they do not scale well as the number of inputs x increases, so called "curse of dimensionality" [157, 2]. For example, for the polynomial case of order M with N inputs, the number of coefficients grow following the power law N^M.

2.2.1.1 Neural Networks

One of the model representations that counters these limitations are *neural networks* [2, 158, 159], also called feedforward neural networks, or multilayer perceptrons (MLPs) [159]. Neural networks are said to be *universal approximators* due to their approximation properties, which are able to approximate any function given enough network size [160, 161, 162, 163, 164, 165, 166, 167, 168]. Originally inspired by biological neural networks [169, 170, 171, 172], neural networks are a series of functional transformations where a linear combination of parameters $\boldsymbol{w}$ are

transformed for nonlinear activation functions $h(\cdot)$ in cascade along *layers*, which consists of many *units* that act in parallel representing a vector-to-scalar function [159]. A neural network with D inputs, one hidden layer with M units, and K outputs, as seen in Figure 2.1, is be represented by the equation

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = h_2\left(\sum_{j=1}^{M} w_{kj}^{(2)} h_1\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(1)}\right). \tag{2.2}$$

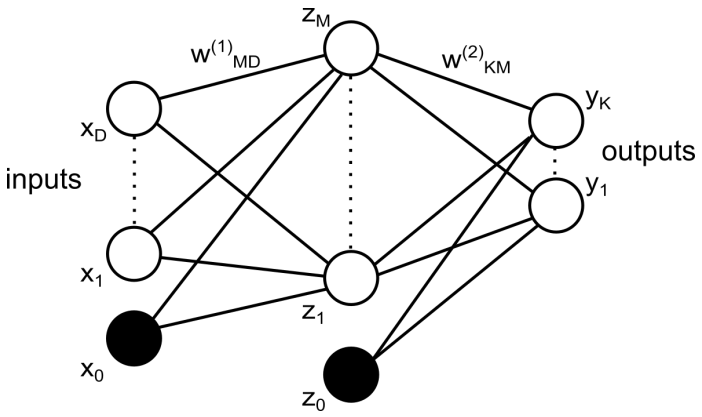


Figure 2.1: A neural network with D inputs, one hidden layer with M units, and K outputs. Adapted from [2].

The activation function $h(\cdot)$ plays an important role in neural networks since it gives nonlinear properties while propagating the inputs and also affects how the the gradients are back-propagated during training. Example of the most common activation functions are: Linear (Figure 2.2), Sigmoid (Figure 2.3), Hyperbolic Tangent (Tanh, Figure 2.4), Rectified Linear Unit (ReLU, Figure 2.5), Leaky Rectified Linear Unit (Leaky ReLU, Figure 2.6), Exponential Linear Unit (Figure 2.7), and Softplus (Figure 2.8). Table 2.1 details the equations and their derivatives and Table 2.2 details advantages and disadvantages for the select of activation functions commonly used in neural networks [173].

Table 2.1: Select activation function equations and their derivatives.

Name	Equation	Derivative
Linear (Fig. 2.2)	h(x) = x	h'(x) = 1
Sigmoid (Fig. 2.3)	h(x) = $\frac{1}{1+e^{-x}}$	h'(x) = h(x)(1 - h(x))
Tanh (Fig. 2.4)	h(x) = $\frac{2}{1+e^{-2x}} - 1$	h'(x) = 1 - h(x)^2
ReLU (Fig. 2.5)	h(x) = $\begin{cases} 0, & \text{for } x < 0. \\ x, & \text{for } x \geq 0. \end{cases}$	h'(x) = $\begin{cases} 0, & \text{for } x < 0. \\ 1, & \text{for } x \geq 0. \end{cases}$
Leaky ReLU (Fig. 2.6)	h(x) = $\begin{cases} \alpha x, & \text{for } x < 0. \\ x, & \text{for } x \geq 0. \end{cases}$	h'(x) = $\begin{cases} \alpha, & \text{for } x < 0. \\ 1, & \text{for } x \geq 0. \end{cases}$
ELU (Fig. 2.7)	h(x) = $\begin{cases} \alpha(e^x - 1), & \text{for } x < 0. \\ x, & \text{for } x \geq 0. \end{cases}$	h'(x) = $\begin{cases} \alpha e^x, & \text{for } x < 0. \\ 1, & \text{for } x \geq 0. \end{cases}$
Softplus (Fig. 2.8)	h(x) = $\ln(1 + e^x)$	h'(x) = $\frac{1}{1+e^{-x}}$

Table 2.2: Select activation function advantages and disadvantages.

Name	Advantages	Disadvantages
Linear	Wide range of outputs, useful when scaling is not desired.	Does not add nonlinearities to inputs, only a linear transformation. Constant derivative, so gradient is not dependent on input when back-propagated.
Sigmoid	Bounded output [0,1]. Smooth derivative.	Saturates and kills gradients due to flat derivative towards the tail. Outputs not zero centered, changing dynamics of gradient descent.
Tanh	Bounded output [-1,1]. Smooth derivative.	Saturates and kills gradients due to flat derivative towards the tail.
ReLU	Simple implementation and inexpensive operation. Shown to accelerate learning on vision-based tasks [174].	Large gradients can shift weights in a way that disables neuron units permanently (never activate), the "dead neuron" problem.
Leaky ReLU	Attempts to correct the ReLU "dead neuron" problem. Simple implementation and inexpensive operation.	Large gradients can shift weights in a way that disables neuron units permanently (never activate).
ELU	Attempts to correct the ReLU "dead neuron" problem. Can output negative values. Strong alternative to ReLU.	Output not bounded for $x > 0$.
Softplus	Similar to ReLU, but with smooth gradient. near zero.	Operation not as inexpensive as ReLU.

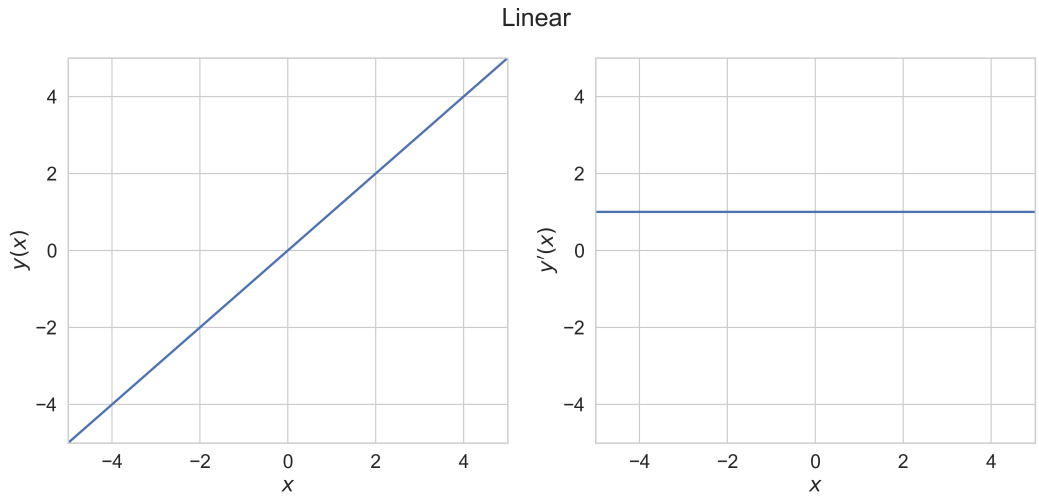


Figure 2.2: Linear activation function $y(x)$ and its derivative $y'(x)$.

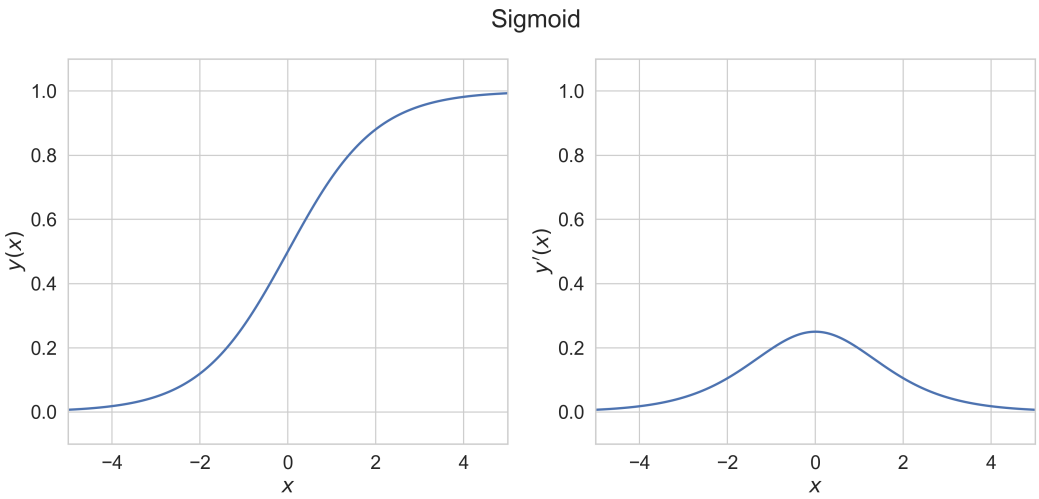


Figure 2.3: Sigmoid activation function $y(x)$ and its derivative $y'(x)$.

2.2.1.2 Learning in Discrete and Continuous Spaces

Learning in neural networks occurs by systematically changing the network weights of each layer in order to drive the predicted output $\hat{y}_i$ closer to the target value or label y_i for the same input sample. This is measured by a *cost function*, generally represented by $\mathcal{L}(\cdot)$, which is different for discrete spaces (classification) in continuous (regression) cases. Common cost function for classification are Cross-Entropy and Hinge loss while for regression we have Mean Squared Error

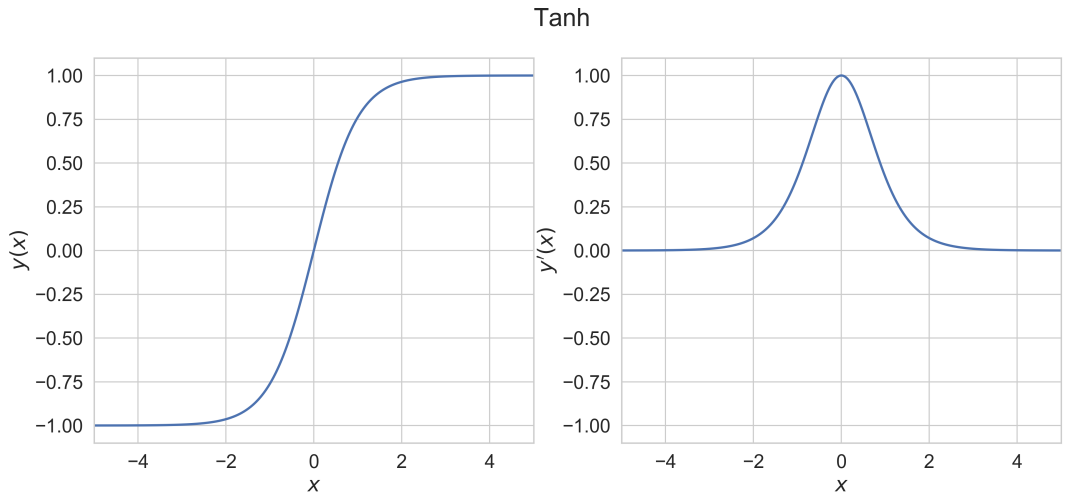


Figure 2.4: Hyperbolic Tangent (Tanh) activation function $y(x)$ and its derivative $y'(x)$.

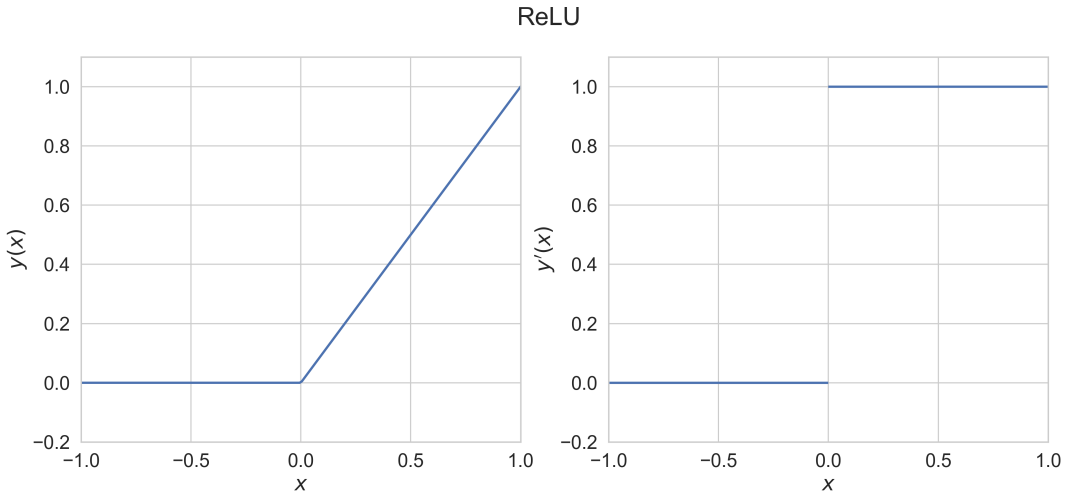


Figure 2.5: Rectified Linear Unit (ReLU) activation function $y(x)$ and its derivative $y'(x)$.

(MSE), Mean Absolute Error (MAE), and Huber loss. The Cross-Entropy (CE) loss, written as

$$\mathcal{L}_{CE}(y_i, \hat{y}_i) = -\frac{1}{N}\sum_{i}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}),$$

is used for classification with C classes. Binary Cross-Entropy (BCE) loss, also known as Log loss, is the special case of the CE loss when the number of classes C is equal to 2 (binary classification

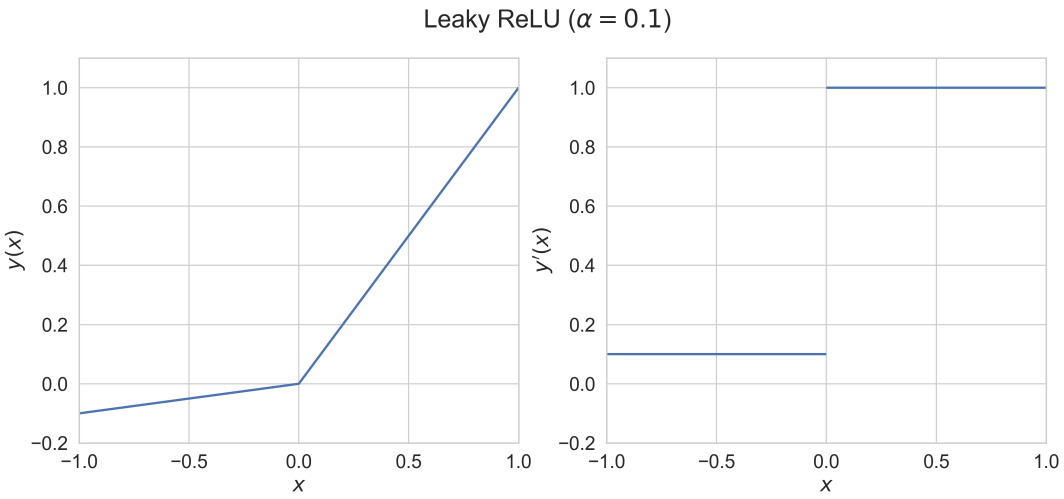


Figure 2.6: Leaky Rectified Linear Unit (Leaky ReLU) activation function $y(x)$ and its derivative $y'(x)$.

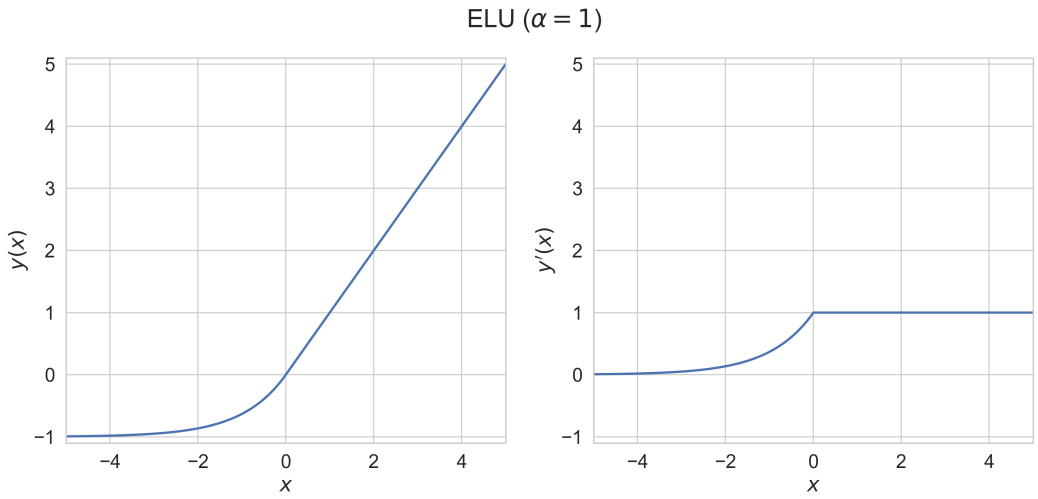


Figure 2.7: Exponential Linear Unit (ELU) activation function $y(x)$ and its derivative $y'(x)$.

problem) and is written as

$$\mathcal{L}_{BCE}(y_i, \hat{y}_i) = -\frac{1}{N}\sum_{i}^{N}\sum_{j=1}^{C=2} y_{ij}\log(\hat{y}_{ij}) = -y_{i1}\log(\hat{y}_{i1}) - (1-y_{i1})\log(1-\hat{y}_{i1}).$$

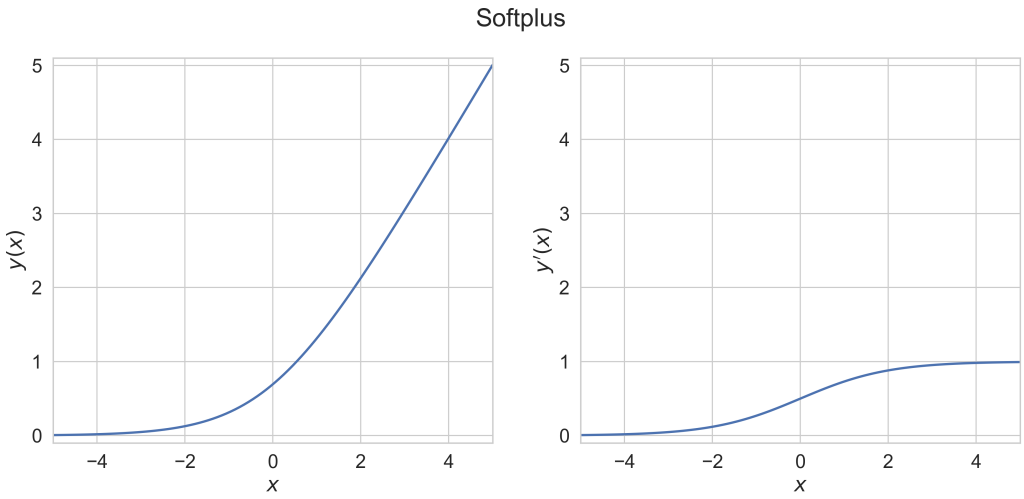


Figure 2.8: Softplus activation function $y(x)$ and its derivative $y'(x)$.

Hinge loss, often used for maximum-margin classification in Support Vector Machines (SVM) [154], is written as

$$\mathcal{L}_{Hinge}(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i}^{N} \max(0, 1 - y_i \hat{y}_i).$$

For regression problems, Mean Squared Error (MSE) loss computes the average of the square of the errors between predicted $\hat{y}_i$ and true y_i value, which results in an arithmetic mean-unbiased estimator:

$$\mathcal{L}_{MSE}(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i}^{N} (y_i - \hat{y}_i)^2.$$

The Mean Absolute Error (MAE) loss, similar to MSE, computes the average of the absolute errors between predicted $\hat{y}_i$ and true y_i value, which results in an median-unbiased estimator:

$$\mathcal{L}_{MAE}(y_i, \hat{y}_i) = \frac{1}{N} \sum_{i}^{N} |y_i - \hat{y}_i|.$$

Huber loss combines the nonlinearity of MSE and linearity of MAE in a single loss function, controlled by the hyperparameter δ, less sensitive to high magnitude error that would normally be

amplified by MSE loss. The Huber loss is written as

$$\mathcal{L}_{Huber}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{for } |y - \hat{y}| \leq \delta. \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2, & \text{otherwise.} \end{cases}$$

Defined a cost function, the neural network weights are updated (also called "trained") using the *backpropagation* algorithm. In backpropagation each weight parameter is computed based on their proportional contribution to the cost for the given input $\boldsymbol{x}$ using the *chain rule* for partial derivatives and gradient descent updates. For example, in the case of the neural network presented in Figure 2.1 in a regression task, the cost or loss would be given by

$$\mathcal{L} = \sum_{j=1}^{K} (a_j^{(L)} - y_j)^2,$$

where L is the index of the hidden layer, K the number of output neurons, and a_j the activation output of the j neuron in the layer. Explicitly, for the case we have three neurons and one bias in the hidden layer, a_j is represented as

$$a_j^{(L)} = h(w_{10}^{(L)} a_1^{(L-1)} + w_{20}^{(L)} a_2^{(L-1)} + w_{30}^{(L)} a_3^{(L-1)} + b_j^{(L)}),$$

where $h(\cdot)$ is a, ideally nonlinear, activation function. Using the chain rule, the loss contribution of each previous activation output can be written as

$$\frac{\partial \mathcal{L}}{\partial a_k^{(L-1)}} = \sum_{j=1}^{K} \frac{\partial a_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial \mathcal{L}}{\partial a_j^{(L)}}.$$

This process is repeated for every previous layer until we reach the input layer and compute $\frac{\partial \mathcal{L}}{\partial x_i}$ and for each weight component $\frac{\partial \mathcal{L}}{\partial w_{ij}}$ along the connection path. Each weight component at time t, denoted w_{ij}^t, is then updated with gradient descent as

$$w_{ij}^{t+1} = w_{ij}^t - \alpha \frac{\partial \mathcal{L}}{\partial w_{ij}^t},$$

where α is a step size hyperparemeter.

2.3 Sequential Decision Making Problems

Mathematically and similarly to any sequential decision making process, in its most general form, the RL problem is formalized as a *Partially Observable Markov Decision Process* (POMDP) [175]. A POMDP $\mathcal{M} = \{\mathcal{S}, \mathcal{O}, \mathcal{E}, \mathcal{A}, \mathcal{T}, r\}$ is characterize by its state-space $\mathcal{S}$ (where a vector of states $\boldsymbol{s} \in \mathcal{S}$), an observation-space $\mathcal{O}$ (where a vector of observations $\boldsymbol{o} \in \mathcal{O}$), an emission probability $\mathcal{E}$ that controls the probability of observing $\boldsymbol{o}$ conditioned to the underlying states $\boldsymbol{s}$, an action-space $\mathcal{A}$ (where a vector of actions $\boldsymbol{a} \in \mathcal{A}$), a transition operator $\mathcal{T}$ (which defines the probability distribution $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$), and the reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (or $r(\boldsymbol{s}, \boldsymbol{a})$), as showed in Figure 2.9 of the previous chapter. In *Fully Observable Markov Decision Process*, or simply a *Markov Decision Process* (MDP), it is assumed full knowledge of the underlying states, removing the dependency to the observations and simplifying the process to $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$.

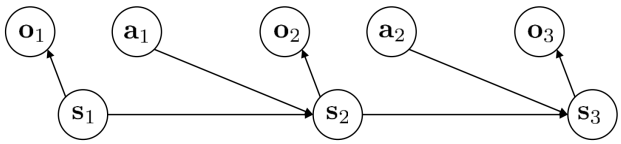


Figure 2.9: Partially-Observable Markov Decision Process diagram. Adapted from [3].

2.4 Cloning Behaviors with Imitation Learning

As discussed in Section 1.2.1.1, *Imitation Learning* provides a directed path to learn demonstrated behaviors by utilizing examples of a secondary policy performing the task, quickly converging to more stable behaviors. This secondary policy can be another agent trained with reinforcement learning, a human, a classical controllers, or any other entity that is able to attempt to solve the desired task.

Behavior Cloning (BC) is the subset of Imitation Learning where the demonstration data is directly used to clone the demonstrated behavior by training a model to generate actions as similar

as possible to the demonstrated data, given similar inputs. In BC, a policy π is trained in order to generalize over a subset $\mathcal{D}$ of states and actions visited during a task demonstration over T time steps:

$$\mathcal{D} = \{\boldsymbol{a}_0, \boldsymbol{s}_0, \boldsymbol{a}_1, \boldsymbol{s}_1, ..., \boldsymbol{a}_T, \boldsymbol{s}_T\}.$$

This demonstration can be performed by a human supervisor, optimal controller, or virtually any other pre-trained policy. In the case of human demonstrations, the human is implicitly trying to maximize what may be represented as an internal reward function for a given task (Equation 2.3), where $\pi^*(\boldsymbol{a}_t^*|\boldsymbol{s}_t)$ represents the optimal policy that is not necessarily known, in which the optimal action $\boldsymbol{a}^*$ is taken at state $\boldsymbol{s}$ for every time step t.

$$\max_{\boldsymbol{a}_0,...,\boldsymbol{a}_T} \sum_{t=0}^{T} r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t=0}^{T} \log p(\pi^*(\boldsymbol{a}_t^*|\boldsymbol{s}_t)) \tag{2.3}$$

Defining the policy of the supervisor as π_{sup} and its estimate as $\hat{\pi}_{sup}$, behavior cloning can be achieved through standard supervised learning, where the parameters θ of a policy π_θ are trained in order to minimize a loss function, such as mean squared error, as shown in Equation 2.4. Behavior cloning can be seen as supervised learning applied to sequential decision making problems.

$$\hat{\pi}_{sup} = \underset{\pi_\theta}{\operatorname{argmin}} \sum_{t=0}^{T} ||\pi_\theta(\boldsymbol{s}_t) - \boldsymbol{a}_t||^2 \tag{2.4}$$

3. FROM SHALLOW TO DEEP REINFORCEMENT LEARNING

This chapter introduces the notation and defines the Reinforcement Learning (RL) problem, including its mathematical formulation as a Markov Decision Process, types of Reinforcement Learning algorithms, and the transition from “Shallow" to “Deep" Reinforcement Learning. Reinforcement Learning is very closely related to the theory of classical optimal control, dynamic programming, stochastic programming, and optimization [176]. However, while optimal control assumes perfect knowledge of the system’s model, RL operates based on performance metrics returned as consequence of interactions with an unknown environment [140].

3.1 Partially and Fully Observable Markov Decision Processes

Mathematically and similarly to the supervised learnig problem presented on Chapter 2, in its most general form, the RL problem is formalized as a *Partially Observable Markov Decision Process* (POMDP) [175]. A POMDP $\mathcal{M} = \{\mathcal{S}, \mathcal{O}, \mathcal{E}, \mathcal{A}, \mathcal{T}, r\}$ is characterize by its state-space $\mathcal{S}$ (where a vector of states $\boldsymbol{s} \in \mathcal{S}$), an observation-space $\mathcal{O}$ (where a vector of observations $\boldsymbol{o} \in \mathcal{O}$), an emission probability $\mathcal{E}$ that controls the probability of observing $\boldsymbol{o}$ conditioned to the underlying states $\boldsymbol{s}$, an action-space $\mathcal{A}$ (where a vector of actions $\boldsymbol{a} \in \mathcal{A}$), a transition operator $\mathcal{T}$ (which defines the probability distribution $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t)$), and the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ (or $r(\boldsymbol{s}, \boldsymbol{a})$), as showed in Figure 2.9 of Chapter 2. In *Fully Observable Markov Decision Process*, or simply a *Markov Decision Process* (MDP), it is assumed full knowledge of the underlying states, removing the dependency to the observations and simplifying the process to $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$.

3.2 The Reinforcement Learning Problem

3.2.1 Problem Definition

In *Reinforcement Learning* (RL), it is desired to train an agent to learn the parameters θ of a *policy* (or *controller*) π_θ, in order to map the partially-observable environment’s *observation* vector $\boldsymbol{o}$ (or *state* vector $\boldsymbol{s}$ in fully-observable environments) to agent *actions* $\boldsymbol{a}$. The performance of the agent is measured by a scalar *reward signal* r returned by the environment (external rewards, r_e)

and/or returned by the agent itself (intrinsic rewards, r_i). At each time step t the reward signal can be computed as the sum of all the extrinsic and intrinsic rewards received $r_t = r_{e_t} + r_{i_t}$. Figure 3.1 illustrates this description as a diagram.

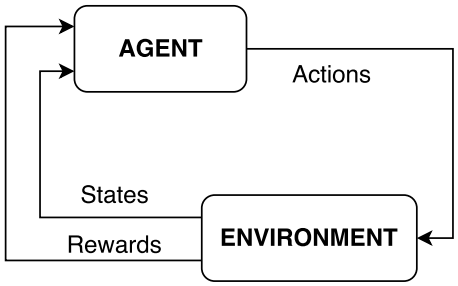


Figure 3.1: Reinforcement learning problem modeled as agent and environment interactions.

This interaction between agent and environment occurs during an *episode*, which is defined as the time interval between the initial time step t_0 and the maximum number of time steps allowed T or until a previously established metric is achieved. At each time step t of a finite time horizon T, the *policy* (or controller) π, parametrized by θ_t, maps the current environment's *states* $\boldsymbol{s}_t$ to *actions* $\boldsymbol{a}_t$: $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$. This action affects the current environment's states $\boldsymbol{s}_t$ which evolves to $\boldsymbol{s}_{t+1}$ based on the environment's transition distribution (dynamics) $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. During an episode, the sequence of states observed and actions taken over a number of time steps T can be represented by a *trajectory* $\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T\}$. The total reward per episode R is defined as the sum of the rewards received for each time step, as shown in Equation 7.2.

$$R = \sum_{t=0}^{T} r_t = \sum_{t=0}^{T} \left(r_{e_t} + r_{i_t}\right). \tag{3.1}$$

Similarly, the expected total reward per episode received by a policy $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ can be defined by Equation 7.3.

$$R_{\pi_\theta} = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi_\theta} [r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)]. \tag{3.2}$$

To simplify further derivation of the algorithm it is assumed a fully-observable environment, where the observation-space $\mathcal{O} = \mathcal{S}$ (state-space) and, consequently, vector of observations $\boldsymbol{o} = \boldsymbol{s}$ (states).

At each time step t of a finite time horizon T, the *policy* (or controller) π, parametrized by θ_t, maps the current environment's *states* $\boldsymbol{s}_t$ to *actions* $\boldsymbol{a}_t$: $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$. This action affects the current environment's states $\boldsymbol{s}_t$ which evolves to $\boldsymbol{s}_{t+1}$ based on the environment's transition distribution (dynamics) $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The environment also returns a scalar *reward function* r that evaluates the action taken $\boldsymbol{a}_t$ at the state $\boldsymbol{s}_t$: $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$.

The probability of experiencing a given trajectory τ (sequence of state $\boldsymbol{s}$ and action $\boldsymbol{a}$ pairs) in a Markov Decision process can be written as:

$$\pi_\theta(\tau) = p_\theta(\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T) \tag{3.3}$$

$$= p(\boldsymbol{s}_1) \prod_{t=1}^{T} \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{3.4}$$

The goal in RL is to find the parameters θ^* that will maximize the objective $J(\theta)$

$$J(\theta) = \mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \tag{3.5}$$

$$= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta(\tau)} [r(\tau)] = \int (\pi_\theta(\tau) r(\tau) d\tau), \tag{3.6}$$

which represents the expected total reward to be received by this policy $\pi_\theta(\tau)$:

$$\theta^* = \operatorname*{argmax}_\theta J(\theta) \tag{3.7}$$

$$= \operatorname*{argmax}_\theta \mathop{\mathbb{E}}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{3.8}$$

To maximize $J(\theta)$, it is possible to apply the gradient operator ∇_θ and compute its gradient

with respect to its parameters θ:

$$\nabla_\theta J(\theta) = \int (\nabla_\theta \pi_\theta(\tau) r(\tau) d\tau). \tag{3.9}$$

Using the following identity

$$\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau) \tag{3.10}$$

on Equation 3.9 it is possible to simplify it to

$$\nabla_\theta J(\theta) = \int (\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau) \tag{3.11}$$

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau) r(\tau)], \tag{3.12}$$

known as the *policy gradient*, which, by substituting τ, can also be written in terms of states and actions over time steps t in a given time horizon T:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t | \boldsymbol{s}_t) \sum_{t=1}^{T} r(\boldsymbol{a}_t, \boldsymbol{s}_t) \right]. \tag{3.13}$$

Since the structure of the objective $J(\theta)$ and its gradient $\nabla_\theta J(\theta)$ are unknown, in practice the only way to evaluate them and approximate the expectation term is by sampling and averaging over N samples:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=0}^{N} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t^{(i)} | \boldsymbol{s}_t^{(i)}) \right) \left(\sum_{t=0}^{T} r(\boldsymbol{a}_t^{(i)}, \boldsymbol{s}_t^{(i)}) \right) \right]. \tag{3.14}$$

The policy is improved by *gradient ascent* according to a step size α, as shown in Equation 4.13 and Figure 3.2, optimizing the agent's policy during the training time in what is called the *vanilla policy gradient* (VPG) in RL.

$$\theta \rightarrow \theta + \alpha \nabla_{\theta} J(\theta) \tag{3.15}$$

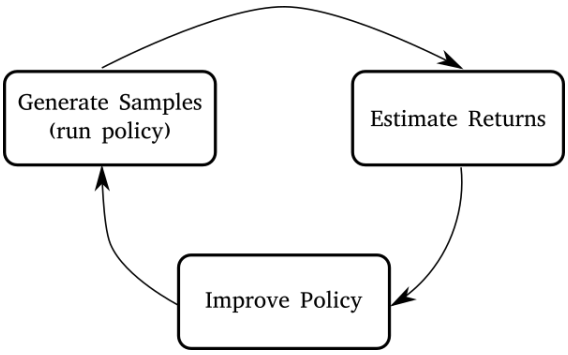


Figure 3.2: Policy evaluation and optimization cycle.

3.2.2 Types of Reinforcement Learning Algorithms

As shown in the previous section, policy gradient methods in RL directly differentiate the objective function, which is to maximize the expected sum of rewards, and improve the policy by following gradient ascent. Other types of RL algorithms directly estimate a *value function* for a policy π of a particular state $V^{\pi}(s)$, representing the expected sum of rewards to be received if starting at that state s, written as

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\pi}[R_t | s_t = s],$$

or the value function of a particular state-action pair $Q^{\pi}(s, a)$, representing the expected sum of rewards to be received if starting at that state s and taking the action a [157], written as

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\pi}[R_t | s_t = s, a_t = a].$$

One can also define an *advantage function* $A(s, a)$ as

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

to quantify the advantage of taking a particular action a at the state s when compared to the average value of that state [177, 61]. Bellman [178, 179] showed that in Markovian Decision Processes there is a relationship between the value of a state and the value of it successor states [157]. After averaging and weighting all probabilities, Bellman showed that "the value of the start state must equal the (discounted) value of the expexted next state, plus the reward expected along the way"[157]:

$$V^{\pi}(s_t) = \underset{\pi}{\mathbb{E}}[r_t + \gamma V^{\pi}(s_{t+1})].$$

We can remove the dependency to the policy π and show that, under the optimal policy, the value of the state $V^*(s_t)$ is equal the expected discount sum of rewards for the best action when the agent is in that state, know as the *Bellman Optimality Equation* [157]:

$$\begin{aligned} V^*(s_t) &= \max_a Q^{\pi^*}(s_t, a_t) \\ &= \max_a \mathbb{E}[r_t + \gamma V^*(s_{t+1})]. \end{aligned}$$

Similarly, the Bellman Optimality Equation can be written for $Q^*(s_t, a_t)$ as

$$Q^*(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a'} Q^*(s_{t+1}, a')],$$

where a' is the action to be taken at state s_{t+1}.

Based on the Bellman equations, other RL algorithms can be derived. A classical example is *Q-learning*, by Watkins [180]. In Q-learning, by interacting with the environment, the agent continuously estimates the state-action value $Q(s_t, a_t)$ as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a_t) - Q(s_t, a_t)],$$

where α is a learning rate hyperparameter.

Another type of RL algorithms called *actor-critic* methods combine both policy gradient and

value function concepts. In Section 3.2.1 it was shown that, in policy gradient methods, the gradient of the objective can be written as

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t | \boldsymbol{s}_t) \sum_{t=1}^{T} r(\boldsymbol{a}_t, \boldsymbol{s}_t) \right]. \tag{3.16}$$

Note that the term $\sum_{t=1}^{T} r(\boldsymbol{a}_t, \boldsymbol{s}_t)$ is unknown until the end of the episode or a given number of time steps T, however it can be estimated by the value function $\hat{Q}_t^\pi$ as

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t | \boldsymbol{s}_t) \hat{Q}_t^\pi \right] \tag{3.17}$$

and work as a *critic* in order to evaluate the actions taken by policy π, the *actor*. Hence, actor-critic algorithms incorporate value learning by updating the estimates of its value function and using it to compute the gradient of the policy.

Policy gradient, value learning, and actor-critic are also classified as *model-free* algorithms because they assume the dynamic model of the environment is unknown to the agent. Other algorithms, called *model-based*, leverage interaction data between agent and environment to learn the dynamics of the environment (or are given the model), which could be use for planning [99], perform policy updates using simulated data [102], and others.

3.3 From Shallow to Deep Representations

As a side note, in Deep RL where deep neural networks are used as learning representation for the policies, a typical policy is represented by a neural network with two fully-connected layers with 64 neurons each (plus a bias term). If the dimension of the input is N and the dimension of the output is M, this leads to $(N+1)*64 + (64+1)*64 + (64+1)*M$ parameters. For example, a policy for a continuous dynamical system with twelve inertial states and three control outputs would be represented by 5187 parameters.

The two key ideas that allow reinforcement learning to achieve the desired goal are: sampling through interactions to compactly represent the dynamics of an unknown stochastic environment,

and the use of function approximations to represent policies [181] that guides the algorithm during the action selection. Recently, due to advances in computational power, deep neural networks, or Deep Learning, is being used for sensory processing and function approximation. Goodfellow et al. [159] define deep learning as the idea of training computers to gather knowledge by experience and to learn complex concepts by building them out of simpler ones. Feeding raw pixels of an image to a deep neural network is possible to see how it combines simple concepts of edges to create concepts of corners and contours, and combine the concepts of corners and contours to represent the concept of an object, as seen in Figure 3.3. In RL, Deep Learning is used to represent polices and value functions (or the dynamics model, in model-based approaches) — so-called Deep Reinforcement Learning (Deep RL) [87, 75]. The higher complexity and plasticity of deep neural networks allows better quality on feature and data representations [159] and is directly related to the performance of reinforcement learning algorithms [75].

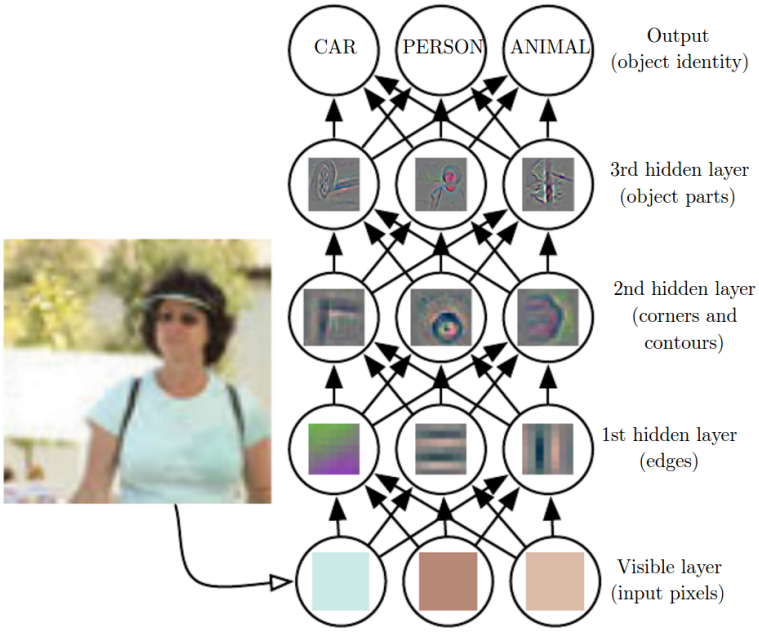


Figure 3.3: Visualization of each layer of a deep neural network use to classify different images. Reprinted from [4].

3.3.1 Inductive Bias

A core concept of learning is to be able to generalize from past experiences (for example, a training dataset) to address unseen situations [182]. Inductive bias refers to “any basis for choosing one generalization over another, other than strict consistency with the observed training instances." [182].

This research work assumes that the human brain and its underlying decision process is nonlinear and stochastic. Nonlinear due to the neural pathway arrangement and connections [183], and stochastic in a sense that different action outputs can be observed from the same observation inputs depending on the current human emotional states [184]. To address this nonlinearity and stochasticity, this research adopts deep neural networks with Gaussian outputs to represent the learning policy. This representation also allows processing the high-dimensional continuous observation and action-space encountered in real-world robotic applications. By using neural networks it is also assumed that the policy to be learned, and the function that approximates it, has a smooth differentiable gradient which can be learned through back-propagation.

3.4 Expanding Q-learning to Continuous Space with Deep Reinforcement Learning

Tabular Q-learning has been a popular algorithm in reinforcement learning due to its simplicity and convergence guarantees [180]. According to Lillicrap et al. [87], it is not possible to directly apply Q-learning to continuous action spaces stating that the optimization is too slow to be practical with large and unconstrained function approximators (required for continuous spaces). The traditional tabular Q-learning is also unfeasible due to the “curse of dimensionality”, when the discretized action space grows exponentially with the number of degrees-of-freedom of the problem [185]. Due to these restrictions, Lillicrap et al. [87] presented a novel model-free, off-policy actor-critic reinforcement learning algorithm using deep neural networks to learn policies in continuous action spaces based on Silver et al. [86] called the Discrete Policy Gradient (DPG) algorithm. It is model-free because it does not require model dynamics to learn the control policy. It is off-policy because it can follow a different control policy than the one that is being learned (can use different

policies to explore the state space). Actor-critic because it has different structures to improve the policy (actor, through the gradient of the policy's performance with respect to its parameters) and evaluate it (critic, through function approximation to estimate expected future rewards).

According to Silver et al. [86], the policy gradient is the gradient of the policy's performance, which on DPG is represented by the actor $\mu(s|\theta^{\mu})$, where θ^{μ} is the actor's network parameters. A critic $Q(s, a|\theta^{Q})$ is learned using the traditional Q-learning approach [180], where θ^{Q} is the critic's network parameters. The actor parameters are updated by applying the chain rule to the start distribution J with respect to the actor parameters θ^{μ} [87], as showed in Eq. (3.18).

$$\begin{aligned} \nabla_{\theta^{\mu}} J &\approx \mathbb{E}[\nabla_{\theta^{\mu}} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t|\theta^{\mu})}] \\ &= \mathbb{E}[\nabla_a Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}] \end{aligned} \tag{3.18}$$

Prior to the work presented by Mnih et al. [75], it was thought that using large nonlinear function approximators to learn critic functions was difficult and unstable [87]. However, using the techniques of replay buffer and target Q networks introduced by Deep Q-Networks (DQN) was possible to train actor-critic methods using neural function approximators. The resulting algorithm, Deep DPG (DDPG), is able to learn competitive policies using only low-dimension observations (e.g. coordinates, velocities, angles, or any other form of sensor reading). The main advantage of DDPG is being able to do off-policy exploration, one of the major challenges when learning on continuous spaces. The exploration policy was constructed by adding process noise $\mathcal{N}$ to the actor policy [87]:

$$\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N} \tag{3.19}$$

The Deep Deterministic Policy Gradient (DDPG) algorithm is outlined in Algorithm 1 below and works as follow: two neural networks are initialized - one to work as the actor $\mu(s|\theta^{\mu})$, and the second to work as the critic $Q(s, a|\theta^{Q})$. Two copies of these networks are created (they will work as the target actor $\mu'(s|\theta^{\mu})$ and the target critic $Q'(s, a|\theta^{Q})$ networks). An empty replay buffer $\mathcal{R}$ is created to store what the agent is experiencing by interacting with the environment (states s,

actions a, and rewards r). Each interaction between the agent and environment count as a time step (represented by t). A collection of a given number of time steps T is called an episode. The DDPG algorithm loops for a desired number of episodes M. In the beginning of each episode, a random process $\mathcal{N}$ is initialized (for state exploration, explained below) and the agent is placed in a different initial state s_1 (from where it has a different initial observation). For each time step an action is sampled from the actor network and added with exploration noise (noisy actions allow the agent to explore the state space beyond its policy, which leads to higher future rewards). This action is executed in the environment, which returns the next state and the reward for the action. The most recent experience (reward, action applied, and the current and next state) is stored in the replay buffer. A batch of N experiences is sampled from the replay buffer to train the learning algorithm. The training consists of the following steps: the critic Q evaluates the sampled states and actions, adds the returned reward, and computes the target value y; the critic network computes a gradient step to update its parameters based on the target value; and the actor network μ is updated based on the critic's network gradient - the policy gradient. The parameters of the actor and critic networks (θ^{μ} and θ^{Q}, respectively) are copied to its respective targets ($\mu'(s|\theta^{\mu})$ and $Q'(s,a|\theta^{Q})$, respectively) based on the target update rate hyperparameter. Following this process, the actor network tends to increase the probability of suggesting actions that will be better evaluate by the critic, which will lead to higher rewards and higher controller performance.

Algorithm 1 Deep Deterministic Policy Gradient (DDPG) [87]

Randomly initialize critic $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ neural networks with weights θ^Q and $\theta^Q\mu$
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $\mathcal{R}$
for episode = 1, M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1, T **do**
        Select action $a_t = \mu(s_t|\theta^u) + \mathcal{N}$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and new state $s_{t=1}$
        Store transition (s_t, a_t, r_t, s_{t+1}) in R
        Sample a random minibatch of N transitions (s_t, a_t, r_t, s_{t+1}) from R
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss, Eq. (10.7)

$$L = \frac{1}{N}\sum_i^N (y_i - Q(s_i, a_i|\theta^Q))^2 \tag{3.20}$$

        Update the actor policy using the sampled policy gradient, Eq. (3.21)

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i^N \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu()s_i} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i} \tag{3.21}$$

        Update target networks, Eq. (3.22)

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \end{aligned} \tag{3.22}$$

4. CASE STUDY: DEEP REINFORCEMENT LEARNING ON INTELLIGENT MOTION VIDEO GUIDANCE FOR UNMANNED AIR SYSTEM GROUND TARGET TRACKING*

Tracking motion of ground targets based on aerial images can benefit commercial, civilian, and military applications. On small fixed-wing unmanned air systems that carry strapdown instead of gimbaled cameras, it is a challenging problem since the aircraft must maneuver to keep the ground targets in the image frame of the camera. Previous approaches for strapdown cameras achieved satisfactory tracking performance using standard reinforcement learning algorithms. However, these algorithms assumed constant airspeed and constant altitude because the number of states and actions was restricted. This paper presents an approach to solve the ground target tracking problem by proposing the Policy Gradient Deep Reinforcement Learning controller. The learning is based on the continuous full-state aircraft states and uses multiple states and actions. Compared to previous approaches, the major advantage of this controller is the ability to handle the full-state ground target tracking case. Policies are trained for three different target cases: static, constant linear motion, and random motion. Results presented in the paper on a simulated environment show that the trained Policy Gradient Deep Reinforcement Learning controller is able to consistently keep a randomly maneuvering target in the camera image frame. Learning algorithm sensitivity to hyperparameters selection is investigated in the paper, since this can drastically impact the tracking performance.

4.1 Problem Definition

Following breakthroughs in artificial intelligence and cost reduction of unmanned air system (UAS) platforms, novel applications combining UAS and computer vision for tracking ground targets are on the rise. Examples of civilian and military applications that directly benefit from autonomous UAS with ground target tracking capabilities are commercial package delivery and disaster relief response equipped with sense-and-avoid (SAA) technology [186, 187], aerial filming

*Adapted with permission from "Deep Reinforcement Learning on Intelligent Motion Video Guidance for Unmanned Air System Ground Target Tracking", by Vinicius G. Goecks and John Valasek, presented at the AIAA Scitech 2019 Forum [5], Copyright 2019 by the American Institute of Aeronautics and Astronautics.

and photography [188], and military intelligence, surveillance, and reconnaissance (ISR) programs [189, 190].

Ground target tracking can be performed by using cameras fixed to the UAS body (camera frame fixed with respect to the aircraft frame) or gimbaled cameras (camera frame independent of the aircraft frame). The problem is simplified by using gimbaled cameras, in which the image frame can be controlled independently of the aircraft trajectory. Unfortunately, small fixed-wing UAS have restricted payload capabilities, which restricts the use of onboard gimbaled cameras. The immediate solution is to equip small fixed-wing UAS with cameras fixed to the fuselage. This requires aircraft maneuvers to keep the target of interest in the image frame.

Current ground target tracking approaches using fixed cameras [191, 192] are able to achieve satisfactory tracking performance using reinforcement learning algorithms to train a control policy on a simulated environment, which is later transferred to the physical hardware. These approaches rely on training the control policy on simplified planar motion simulated environments, with constant altitude and airspeed, restricting the controller to a single-input to the aircraft due to limitations of the learning algorithm applied.

This research builds upon the previously presented work [191, 193] by proposing the development of a learning controller that handles the full-state continuous target tracking case by incorporating concepts of deep reinforcement learning (Deep RL) and policy gradient (PG) algorithms. The main contributions of this research are i) development of a learning controller that handles the full-state continuous target tracking case by incorporating concepts of deep reinforcement learning (Deep RL) and policy gradient (PG) algorithms; ii) investigation of different learning controller designs to achieve better performance on the tracking problem; and iii) comparison of the tracking performance between previous work that relied on discretization of the state-space and current work the uses the full-state continuous state-space. The PG Deep RL controller uses the 6-Degree-of-Freedom (DOF) linear aircraft simulation to extract target position on the image frame. The aircraft states consist of linear velocities and position; roll, pitch, and yaw attitude angles and rates. The controls consist of body-axis pitch, roll and yaw rates; throttle and nozzle position.

4.2 Related Work

Valasek et al. [191] developed a machine learning algorithm approach for learning control policies based on target position in the image frame (pixel position) and current aircraft roll angle. They developed both Q-learning and Q-learning-with-eligibility-traces policies to solve the tracking problem. Dunn and Valasek [193] developed a similar approach to localize atmospheric thermal locations and then guide an UAS to soar from one to another. This research was inspired on how they framed the target tracking problem and it also builds upon their camera and target dynamics.

Noren et al. [192] detailed the initial flight testing of a previously trained control policy [191] for the autonomous tracking of ground targets. Their work showed that it is possible to train a controller using machine learning algorithms by using a simulated environment and later deploy the learned policy to hardware. Even limited by a planar motion simulation and single-output controller, they were able to achieve satisfactory tracking performance on the tracking problem when flying the algorithm trained on a simulated environment. The authors expected to follow the same development route presented by [192] and deploy the presented Deep RL PG algorithm to hardware.

There were also alternative approaches besides machine learning algorithms to solve the target tracking problem with a fixed strap-down camera on a small UAS. Beard and Egbert [194] derived a explicit roll-angle and altitude-above-ground-level (AGL) constraints for road tracking that that guarantees the target will remain in the camera frame. Beard and Saunders [195] later extended their previous work proposing a non-linear guidance law using range-to-target and bearing-to-target information obtained from target motion in the image plane to plan trajectories for the aircraft. Both approaches are validated on simulation and flight testing.

Advances in modern Deep RL algorithms greatly contributed to the presented research. Schulman et al. presented a series of improvement to classic policy gradient algorithms in order to make them tractable and suitable to optimize policies represented by deep neural networks. Trust Region Policy Optimization (TRPO) [66] and Proximal Policy Optimization (PPO) [67] both rely on restricting the gradients during the policy update, either by a trust region (defined according to

the Kullback-Leibler divergence between current and previous policy) or by a clipped objective (defined by the log probability ratio of current and previous policy). Both approaches are able to stabilize learning of policies represented by deep neural networks, although PPO leads to faster results in practice. Schulman et al. also reduced the variance of the policy gradient estimates by proposing a exponentially-weighted estimator, known as Generalized Advantage Estimation (GAE) [196]. These three insights were incorporated to the present research and greatly improved the learning performance.

The main weakness of the presented approach is it increase computational complexity. Even if the control policy is not update in real-time during flight, the policy is still represented by deep neural networks, which require more computational when compared to classical controllers. The current approach also requires extensive validation and verification, since there is no formal proof of convergence of the learning algorithm. Additionally, hard constraints should be added to aircraft states and controls in order to guarantee operation within safety limits.

4.3 Learning Tracking Policies with Reinforcement Learning

The proposed PG Deep RL controller frames ground target tracking as a reinforcement learning problem. In Reinforcement Learning (RL), it is desired to train an agent to learn the parameters θ of a *policy* (or *controller*) π_θ, in order to map the partially-observable environment's *observation* vector $\boldsymbol{o}$ (or *state* vector $\boldsymbol{s}$ in fully-observable environments) to agent *actions* $\boldsymbol{a}$. The performance of the agent is measured by a scalar *reward signal* r returned by the environment. Figure 3.1 illustrates this description as a diagram and Figure 4.1 shows how it can be applied to the target tracking case.

In the target tracking case proposed by this work, the environment comprises the aircraft, camera, and ground target model, explained in more details in the following paragraphs. Environment states are represented by the aircraft states (linear position and velocity, roll, pitch, yaw angles and rates) and target pixel position in the image plane — see Equation 4.1. The agent is represented by the PG Deep RL controller, explained in more details in Section 4.4. The agent controls the aircraft through elevator, throttle, nozzle position, aileron, and rudder actions, δ_e, δ_T, δ_n, δ_a, δ_r respectively — see Equation 4.2. Rudder control was removed to reduce complexity of the model for the PG Deep RL

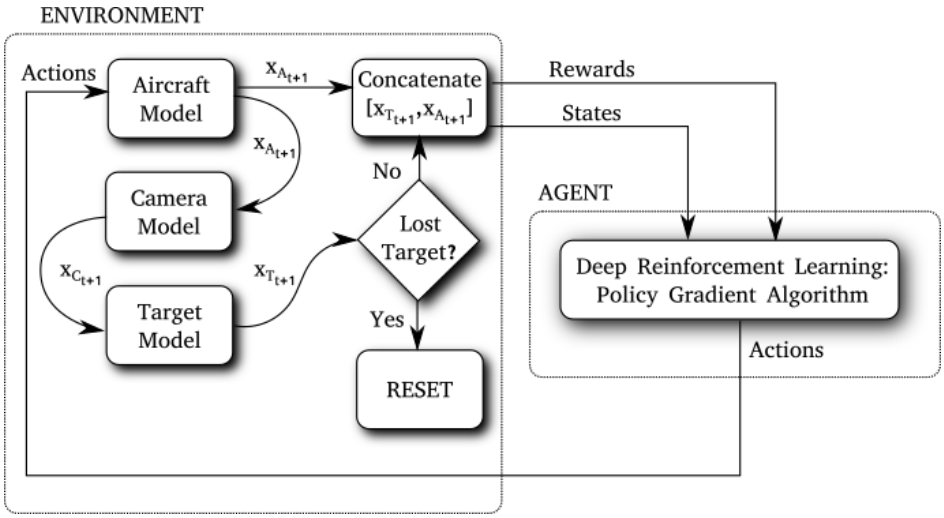


Figure 4.1: Intelligent motion video guidance for unmanned air system ground target tracking modeled as a reinforcement learning problem. Reprinted from [5].

controller. The agent's actions are evaluated by the a scalar value, the reward signal. The reward signal is computed based on the normalized Euclidean distance of the target to the center of the image plane — see Equation 4.3.

$$\mathcal{S} = \{X_T, Y_T, u, w, q, \theta, x, z, v, p, r, \phi, \psi, v\} \tag{4.1}$$

$$\mathcal{A} = \{\delta_e, \delta_T, \delta_n, \delta_a, \delta_r\} \tag{4.2}$$

$$r = \left(1 - \frac{\sqrt{X_T^2 + Y_T^2}}{\sqrt{X_{TMAX}^2 + Y_{TMAX}^2}}\right)^2 \tag{4.3}$$

The PG Deep controller optimizes the action selection by trial-and-error process on a simulated environment. The simulation is broke down in episodes, starting with the aircraft on a random position in the inertial space and ground target initially in the image frame. The episode resets if the target leaves the image frame or if aircraft performs any maneuver outside the safety limits, manually defined by the controller design.

4.4 Policy Gradient Deep Reinforcement Learning Controller

This section details how each part is modeled and how the tracking policies are learned.

One of the main contributions of this research is the development of a learning controller that handles the full-state continuous target tracking case, which includes the full-state continuous simulation of the aircraft, camera, and target. The aircraft is represented by a linear model of the AV-8B Harrier — Figure 4.2 illustrates the aircraft, Equations 4.4 and 4.5 represent the longitudinal and lat/d model, respectively, and Table 4.1 its trim parameters. Camera specifications are shown in Table 4.2. The target is modeled as a point mass with planar motion on the XY plane. Policies are trained for three different target cases: static, constant linear motions, and random motion.

Figure 4.2: Illustration of AV-8B Harrier, whose linear model is used to validate the proposed PG Deep RL controller. Reprinted from [5].

$$\begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \\ \dot{x} \\ \dot{z} \end{bmatrix} = [A_{LON}] \begin{bmatrix} u \\ w \\ q \\ \theta \\ x \\ z \end{bmatrix} + [B_{LON}] \begin{bmatrix} \delta_e \\ \delta_T \\ \delta_n \end{bmatrix}, \tag{4.4}$$

where

$$[A_{LON}] = \begin{bmatrix} -0.0693 & -0.0006 & -0.132 & -32.171 & 0 & 0 \\ 0.0199 & 0.0005 & 10.1644 & -0.3936 & 0 & 0 \\ 0 & 0 & -0.0409 & 0.0373 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and

$$[B_{LON}] = \begin{bmatrix} 3.816 & 0.0237 & -32.667 \\ 4.255 & -1.008 & -0.724 \\ -4.648 & -0.0094 & -0.0679 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

$$\begin{bmatrix} \dot{v} \\ \dot{p} \\ \dot{r} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{y} \end{bmatrix} = [A_{LATD}] \begin{bmatrix} v \\ p \\ r \\ \phi \\ \psi \\ y \end{bmatrix} + [B_{LATD}] \begin{bmatrix} \delta_a \\ \delta_r \end{bmatrix} \tag{4.5}$$

where

$$[A_{LATD}] = \begin{bmatrix} 0 & 0.131 & -10.252 & 32.171 & 0 \\ 0 & -0.105 & 0.0264 & 0 & 0 \\ 0 & -0.0027 & -0.0489 & 0 & 0 \\ 0 & 1 & 0.0122 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and

$$[B_{LATD}] = \begin{bmatrix} -1.0263 & 4.972 \\ 13.735 & 0.454 \\ 0.959 & -1.347 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Table 4.1: Aircraft trim parameters. Reprinted from [5].

Trim Parameter	**Value**
M_1	0.009
U_1 (ft/s)	10
H_1 (ft)	100
δ_e (deg)	1.82
δ_n (deg)	88
δ_T (deg)	89.5

Reinforcement Learning is concerned about learning in an unknown stochastic environment and can be formalized as a *Partially Observable Markov Decision Process* (POMDP) [175]. A POMDP $\mathcal{M}$ can be characterize by its state-space $\mathcal{S}$ (where a vector of states $\boldsymbol{s} \in \mathcal{S}$), an observation-space $\mathcal{O}$ (where a vector of observations $\boldsymbol{o} \in \mathcal{O}$), an action-space $\mathcal{A}$ (where a vector of actions $\boldsymbol{a} \in \mathcal{A}$), a transition operator $\mathcal{T}$ (which defines the probability distribution $p(\boldsymbol{o}_{t+1}|\boldsymbol{o}_t)$), and the reward function

Table 4.2: Modeled camera specifications. Reprinted from [5].

Parameter	Value
Resolution (pixels)	1024x768
Aspect Ratio	4:3
Horizontal Field of View (deg)	90
Vertical Field of View (deg)	30
Pan Angle w.r.t. Aircraft Frame (deg)	-90
Tilt Angle w.r.t. Aircraft Frame (deg)	-20

$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (or $r(\boldsymbol{s}, \boldsymbol{a})$. This definition is illustrated by Equation 4.6 and Figure 2.9:

$$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, r\}. \tag{4.6}$$

To simplify further derivation of the algorithm it is assumed a fully-observable environment, where the observation-space $\mathcal{O} = \mathcal{S}$ (state-space) and, consequently, vector of observations $\boldsymbol{o} = \boldsymbol{s}$ (states).

In Reinforcement Learning, at each time step t of a finite time horizon T, a *policy* (or controller) π, parametrized by θ_t, maps the current environment's *states* $\boldsymbol{s}_t$ to *actions* $\boldsymbol{a}_t$: $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$. This action affects the current environment's states $\boldsymbol{s}_t$ which evolves to $\boldsymbol{s}_{t+1}$ based on the environment's transition distribution (dynamics) $p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The environment also returns a scalar *reward function* r that evaluates the action taken $\boldsymbol{a}_t$ at the state $\boldsymbol{s}_t$: $r(\boldsymbol{s}_t, \boldsymbol{a}_t)$.

During an episode, the sequence of states observed and actions taken over a number of time steps T can be represented by a *trajectory* τ:

$$\tau = \{\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T\} \tag{4.7}$$

The probability of experiencing a given trajectory τ in a Markov Decision process can be written

as:

$$\pi_\theta(\tau) = p_\theta(\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \ldots, \boldsymbol{s}_T, \boldsymbol{a}_T) \tag{4.8}$$

$$= p(\boldsymbol{s}_1) \prod_{t=1}^{T} \pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) \tag{4.9}$$

The goal in RL is to find the parameters θ^* that will maximize the objective $J(\theta)$, which represents the expected total reward to be received by this policy $\pi_\theta(\tau)$:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, J(\theta) \tag{4.10}$$

$$= \underset{\theta}{\operatorname{argmax}} \underset{\tau \sim p_\theta(\tau)}{\mathbb{E}} [\sum_{t=1}^{T} r(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{4.11}$$

To maximize $J(\theta)$, it is possible to compute its gradient with respect to its parameters θ. Since the structure of the objective $J(\theta)$ and its gradient $\nabla_\theta J(\theta)$ are unknown, in practice the only way to evaluate them and approximate the expectation term is by sampling and averaging over N samples:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} [(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\boldsymbol{a}_t^{(i)}|\boldsymbol{s}_t^{(i)}))(\sum_{t=1}^{T} r(\boldsymbol{a}_t^{(i)}, \boldsymbol{s}_t^{(i)}))] \tag{4.12}$$

The policy is improved by *gradient ascent* according to a step size α, as shown in Equation 4.13 and Figure 3.2, optimizing the agent's policy during the training time.

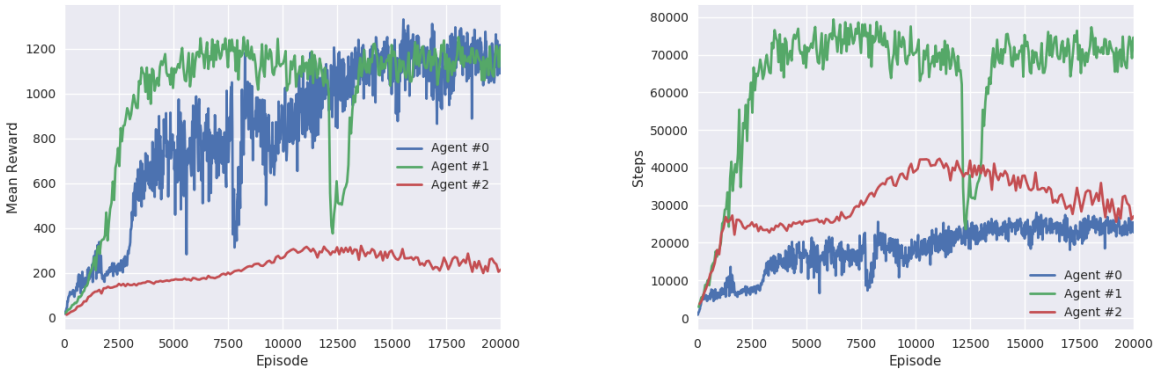
$$\theta \rightarrow \theta + \alpha \nabla_\theta J(\theta) \tag{4.13}$$

4.5 Numerical Results

Machine learning algorithms, specially reinforcement learning using deep neural network as model representation, require thousands or even millions of training episodes to converge to a satisfactory policy. These learning algorithms are also sensitive to hyperparameter values, which tune the learning performance. Due to this reason, it is a good practice to performance a

hyperparameter search before running long-term experiments. During a hyperparameter search, the same algorithm is tested for a short number of episodes with multiple hyperparameter values. Since, intuitively, more data leads to better performance in machine learning, it was decided to investigate the impact of the batch size hyperparameter, which controls the number of episodes that are used to compute the gradient ascent step during the policy improvement phase.

Figure 4.3 shows the reward values and maximum number of steps achieved per episode for different learning agents with different batch sizes, as summarized in Table 4.3. A moderate batch size of 60, compare to 20 and 100, lead to longer tracking time as faster reward convergence. Counter-intuitively, larger batch size does not necessarily leads to faster convergence of the algorithm. Figure 4.4 shows the mean discounted reward achieved during a training batch, averaged by the batch size. Larger batch sizes seems to lead to high variance performance.

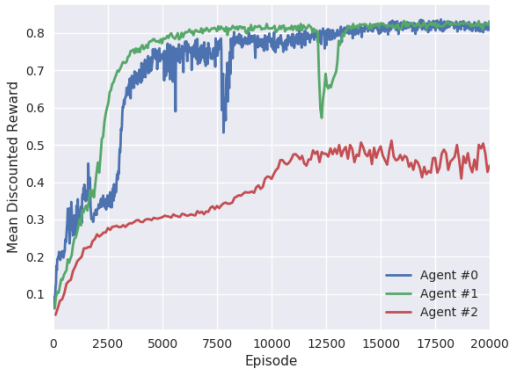


(a) Mean reward achieved during a training batch.

(b) Maximum number of steps during a training batch.

Figure 4.3: Performance evaluation with respect to reward values and maximum number of steps achieved for different learning agents with different hyperparameters (see Table 4.3) during each training episode. Reprinted from [5].

After the hyperparameter search, it was desired to evaluate the tracking performance of the best performing agent. Using the hyperparameters of Agent #1, the agent was retrained during 50000 episodes. The main goal was to keep the target in the image frame throughout the simulation period



(a) Mean discounted reward achieved during a training batch.

(b) Standard deviation of discounted reward achieved during a training batch.

Figure 4.4: Performance evaluation with respect to mean discounted reward and its standard deviation for different learning agents with different hyperparameters (see Table 4.3) during each training episode. Reprinted from [5].

Table 4.3: Hyperparameter values used for the policy gradient deep reinforcement learning algorithm. Reprinted from [5].

Hyperparameter	**Agent #0**	**Agent #1**	**Agent #2**
Number of Episodes per Training Batch	20	60	100
Number of Episodes	20000	20000	20000
Discount Factor γ	0.995	0.995	0.995
Generalized Advantage Estimation λ	0.98	0.98	0.98
KL Divergence Target Value	0.003	0.003	0.003

(one minute). Figure 4.5 shows the the PG Deep RL controller is able to consistently keep the target on the image frame throughout the maximum simulated time after 23,000 training episodes.

4.6 Summary

This paper presented an approach to solve the ground target tracking problem by proposing the Policy Gradient Deep Reinforcement Learning controller. The major advantage of the proposed controller with respect to previous approaches is being able to handle the full-state ground target tracking case, learning based on the full-state continuous aircraft states and controlling multiple outputs. Results on a simulated environment show that, after trained, the controller is able to

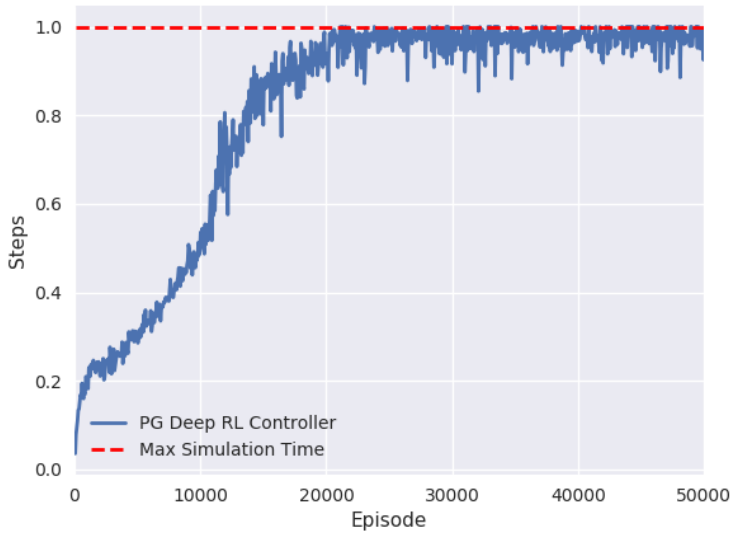


Figure 4.5: Consistent tracking performance of PG Deep RL agent on a simulated environment. Reprinted from [5].

consistently keep the target in the image frame.

Current results shows that the performance of the controller is drastically affected by the choice of the agent's hyperparameters, specifically the number of episodes per training batch — batch size. Contrary to intuition, increasing batch size doesn't lead to improve controller performance. Batch size in the order of 60 episodes leads to faster learning and improved tracking time when compared to 20 and 100 episodes per batch.

After selected the best performing hyperparameters, current results shows that after 23,000 episodes of training the controller is able to learn how to maneuver the aircraft and keep the target in the image frame during the whole simulation period (60 seconds).

5. CASE STUDY: CONTROL OF MORPHING WING SHAPES WITH DEEP REINFORCEMENT LEARNING*

Traditional model-based feedback control techniques are of limited utility for the control of many shape changing systems due to the high reconfigurability, high dimensionality, and nonlinear properties of the plant and actuators of these systems. Computational intelligence and learning techniques offer the promise of effectively leveraging the use of both smart materials and controls for application in aerospace systems such as morphing air vehicles. This paper addresses the challenge of controlling morphing air vehicles by developing a deep neural networks and reinforcement learning technique as a control strategy for shape-memory alloy (SMA) actuators in the context of a morphing wing. The control objective is to minimize the error between an objective and the actual airfoil trailing edge deflection. The proposed controller is evaluated on a simple inverted pendulum for validation, on a 3D printed wing section that is actuated by a composite SMA actuator in a wind tunnel, and on a simulation based on wind tunnel data. Results show that the learning algorithm is capable of learning how to morph the wing. It is also able to control shape changes from arbitrary initial shapes to arbitrary goal shapes using the same trained learning algorithm. The results provide a proof of concept for the use of learning algorithms to control more complex morphing aircraft with continuous states and actions for the outer mold line configuration.

5.1 Problem Definition

Aircraft design is typically a performance trade-off between different flight conditions while satisfying certain safety requirements [197]. Morphing aircraft have the potential to allow an aircraft to change its shape and optimize performance for different mission objectives [198]. Recent efforts in the development of shape-memory alloy (SMA) actuators [199] have shown that SMA actuators enable small-scale camber morphing that can increase performance and versatility for different

*Adapted with permission from "Control of Morphing Wing Shapes with Deep Reinforcement Learning", by Vinicius G. Goecks, Pedro B. Leal, Trent White, John Valasek, and Darren J. Hartl, presented at the 2018 AIAA Information Systems-AIAA Infotech@ Aerospace [6], Copyright 2018 by the American Institute of Aeronautics and Astronautics.

mission objectives [200, 198]. However, these actuators have strong thermomechanical coupling, hysteretic dynamical behavior, and can change response characteristics over time [201]. Traditional model-based feedback control techniques are thus of limited utility especially when multi actuation is considered, however, computational intelligence and learning techniques offer the promise of effectively synthesizing robust and reliable controllers for systems with smart materials.

Academic interest in morphing structures has existed for decades [197], and federal agencies such as NASA have shown strong interest over the years. An example of this is the Mission Adaptive Digital Composite Aerostructure Technologies (MADCAT) [202]; a NASA program to develop a novel aerostructure concept that takes advantage of emerging digital composite materials and manufacturing methods. The objective is to build high stiffness-to-density ratio, ultra-light structures that will facilitate the design of adaptive and aerodynamically efficient air vehicles. Another explored concept was the Spanwise Adaptive Wing Concept [203] that introduced a folding surface at the outboard part of the wing. According to flight conditions, the surface is folded to reduce drag, increase lift, and improve lateral-directional stability. This would allow an increase of performance during taxiing, takeoff, cruise, and supersonic flight.

Considering morphing wings, they not only present challenges from a material science and structure standpoint, but also from a flight control design standpoint due to changing dynamics and inherent uncertainties. A solution proposed by Valasek et al. [136, 137] is the Adaptive-Reinforcement Learning Control technique (A-RLC) that learns near optimal shape changes from arbitrary initial configuration to other arbitrary configuration that maximizes the performance of the aircraft for a given flight condition or maneuver. At the same time, it uses Structured Adaptive Model Inversion (SAMI) as a trajectory tracking controller to handle the time-variant properties, parametric uncertainties, and disturbances. The control law learns how to shape the airfoil by changing two discrete degrees-of-freedom: thickness and camber. The goal of the learning algorithm is to meet nominal aerodynamic goals in terms of lift [204]. The work was later extended to handle four degrees-of-freedom while morphing: wing tip chord, root chord, span, and leading edge sweep angle [205].

This paper addresses the camber control of morphing wings using modern reinforcement learning techniques and deep neural networks. The main advantage of this method is the use of continuous states and actions for the outer mold line (OML) configuration and control inputs. The same learning algorithm is validated in different cases of a pendulum upswing task (different mass, gravity, length) to validate that a nominal learning algorithm can learn a satisfactory policy even when the model changes, which will happen to a morphing airfoil. The validated algorithm is then applied to morph a scaled wing prototype with embedded SMA actuators tested in a wind tunnel environment, and on a simulated morphing wing model created based on the wind tunnel data. Performance of the algorithm is evaluated by commanding different desired airfoil trailing edge deflections from different initial states.

5.2 Learning Morphing Between Wing Shapes

In this paper, the configuration of the wing is defined by the deflection of its trailing edge. The goal of the learning algorithm is to adapt the deep neural network parameters that represent the controller to successfully map the input deflections to output voltages necessary to move from any arbitrary wing configuration to another. The Deep Reinforcement Learning controller receives sensor readings that indicate the current deflection of the wing and a reward signal based off of an evaluation of the control applied. The output voltage is used to heat up the SMA wires via Joule effect leading to austenite transformation. Consequently, the transformation strain is recovered and the wing morphs. Figure 5.1 shows how the learning algorithm interfaces with the morphing wing model, stores past experiences in a memory buffer, maps current states to actions (learns the policy), and computes the gradients based on the received rewards to update the network parameters, as explained in Algorithm 1 and Subsection 3.4.

As mentioned in Section II.B, the network architecture consists of neurons organized as hidden layers. The neural network herein implemented is depicted in Figure 5.2 which in addition shows how these neurons are connected between layers, the specific activation functions used after each layer output, and additional operations to transform the network output. The learning algorithm is evaluated based on the control effort required to morph, time to go from the initial to the final

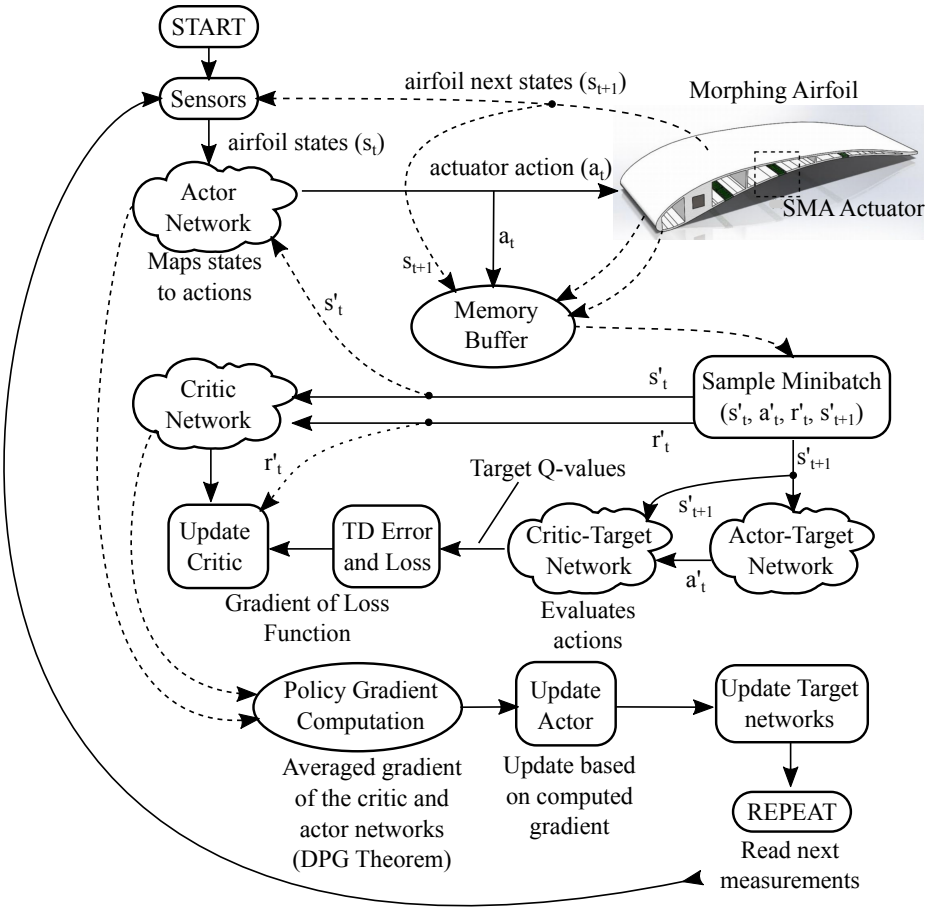


Figure 5.1: Diagram of the learning algorithm and interface with the morphing wing model. Reprinted from [6].

configuration, and number of iterations required to train the learning algorithm to achieve the best performance.

Another important feature of the DDPG algorithm are the hyperparameters. These parameters are used for extra tuning of the learning algorithm and direct affect its learning performance. The hyperparameter values used in this work are depicted in Table 5.1. All the parameters and their function are herein described. The Target Update Rate defines how often (in episodes) the target network parameters are copied from the original network. Target Update Factor is a relaxation factor for the copied network parameters (θ). Actor and Critic Learning Rates define the step size of the gradient update when performing the optimization of the network parameters. Memory Buffer Size defines the number of past experiences (rewards, states, and actions observed from the interaction

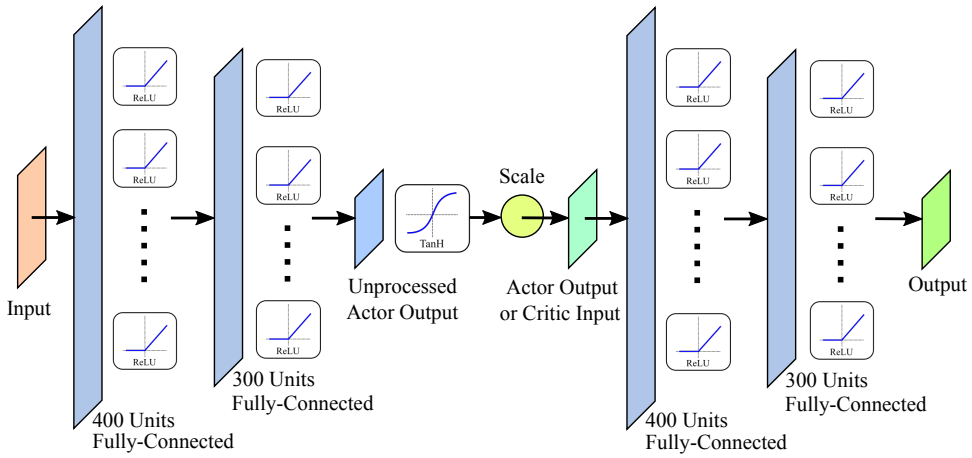


Figure 5.2: Deep neural network architecture that maps the current wing configuration to control inputs. Reprinted from [6].

between agent and environment) that are stored and mixed with current experiences. Minibatch Size controls the number of experiences used during each gradient update on the networks. Discount Factor, bounded between zero and one, defines how future expected rewards are accounted during the learning phase.

Table 5.1: Hyperparameter values used for DDPG algorithm. Reprinted from [6].

Hyperparameter	**Value**
Target Update Rate (Episodes)	100
Target Update Factor (τ)	0.001
Actor Learning Rate	0.0001
Critic Learning Rate	0.001
Memory Buffer Size	100,000
Minibatch Size	64
Discount Factor γ	0.99

5.3 Experimental Setup

The main experimental setup comprised of wind tunnel testing a 3D-printed, avian-inspired, wing prototype actuated by a composite SMA actuator powered by an external power supply, as

seen in Figures 5.3, 5.4, and 5.5. The actuator consists of a PDMS matrix with embedded SMA wires. Trailing edge deflection of the wing, voltage applied to the SMA wire, and its temperature were measured during the experiment.

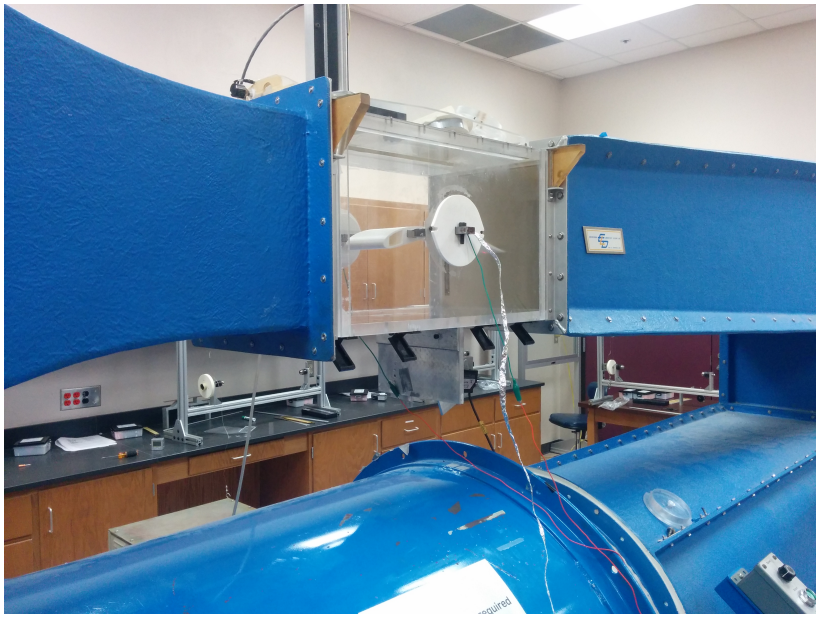


Figure 5.3: Experimental setup for the learning algorithm: prototype in the wind tunnel. Reprinted from [6].

The experiment consists in defining a random setpoint and initial position for the trailing edge of the wing. The learning algorithm reads the current deflection and setpoint applied voltage to the SMA wire in order to morph the wing to the desired state in less than 200 measurement iterations that comprise the learning episode. At the beginning of each episode, the goal and initial states are randomized and the learning algorithm has to perform the task again. During each episode, the learning algorithm is trying to maximize the objective function R ("total reward" of the episode) dependent on the magnitude of control applied and error between current and desired deflection given as:

$$R = \frac{1}{T}\sum_{t=0}^{T} e^{(x^*-x)^2-u/8}, \tag{5.1}$$

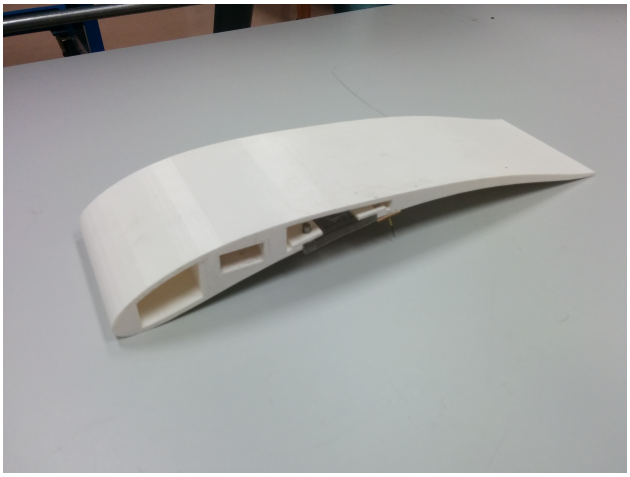


Figure 5.4: Experimental setup for the learning algorithm: top view of the prototype. Reprinted from [6].

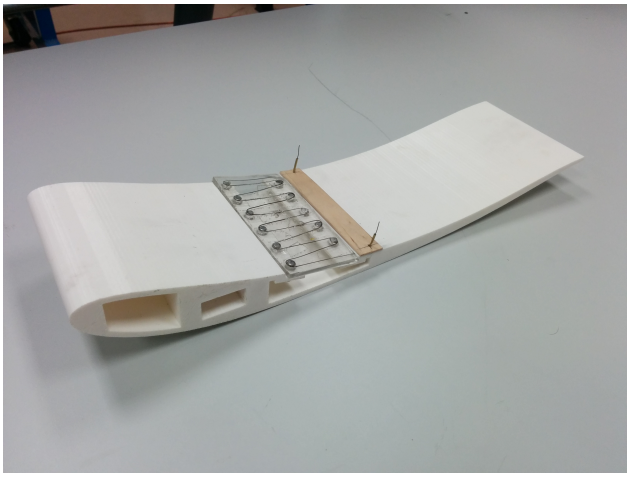


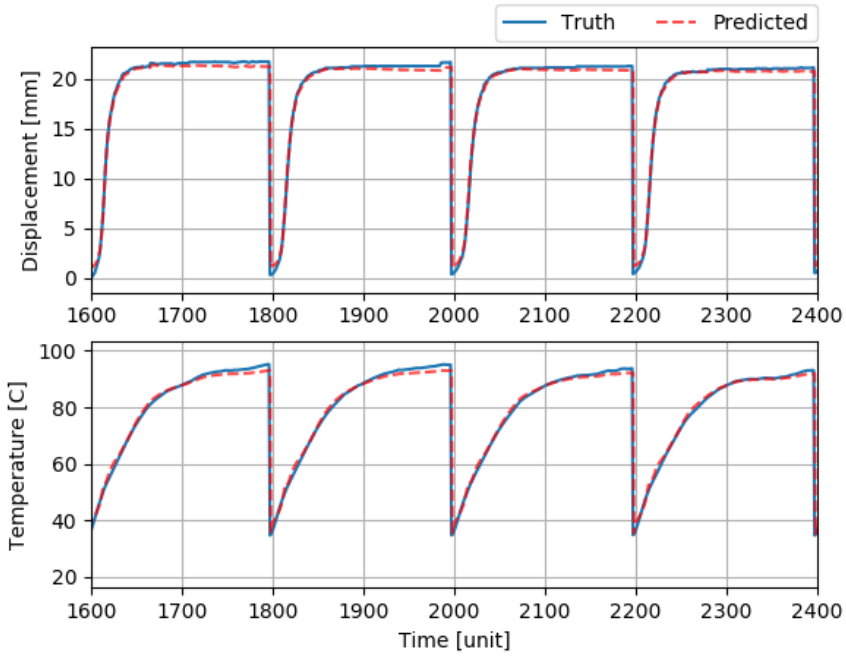
Figure 5.5: Experimental setup for the learning algorithm: bottom view of the prototype. Reprinted from [6].

where T is the total number of time steps t, x is the current deflection, x^* the setpoint, and u is the voltage applied being normalized by its maximum value of 8 volts.

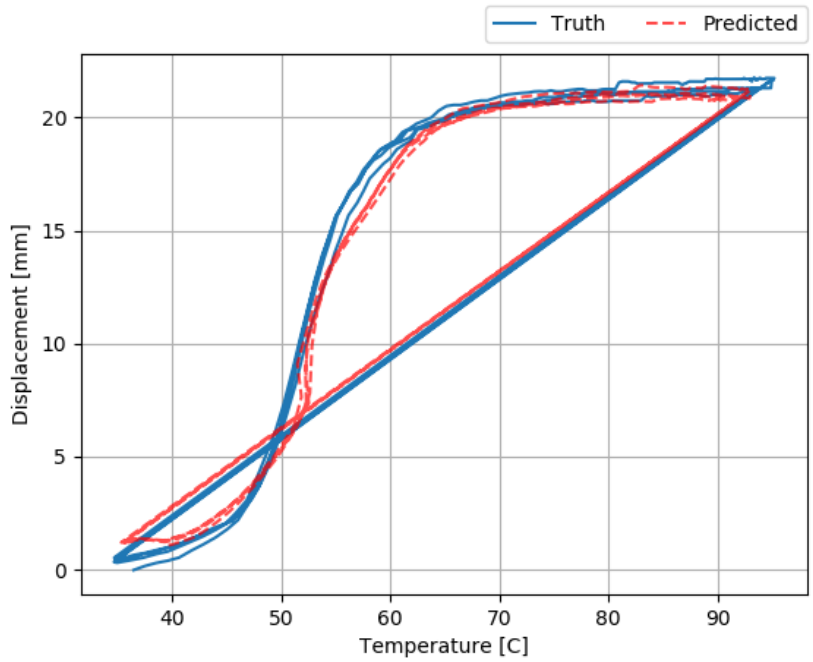
The network used to learn the dynamics comprises of three fully connected layers (each input is connected to every parameter of that layer) with 220, 160, and 130 neurons, respectively. The fully connected layers are followed by dropout connections (randomly disconnects 20% of the units and its input connections during the training phase for better generalization [206]). The

connection between each layer is made through rectified linear units (ReLU) activation functions, which discards the input if it is negative and adds nonlinearities to the network when more hidden layers are stacked. The deep neural network if optimized using the "Adam" (Adaptive Momentum Estimation) [207] optimizer based on a standard mean squared error loss function of the predicted and true temperature and displacement values. "Adam" is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. It is computationally efficient, little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. It stores exponentially decaying average of past squared gradients and past gradients [207].

Applying data-driven learning algorithms directly to hardware is challenging. Deep Reinforcement Learning algorithms are well known to heavily depend on multiple graphic and central processing units (GPUs and CPUs) [75, 208, 78, 71] and training time on the order of days [75, 94] to achieve meaningful results. Simple continuous tasks (e.g., controlling a two-link robotic arm), requires on average more than 2.5 million steps [87] and more complex tasks (e.g., control of a humanoid robot or robots with multiple degrees of freedom), requires on average 25 million steps using different reinforcement learning algorithms [69]. Since it was infeasible to perform a wind tunnel test in the order of weeks, it was desired to create a high-fidelity simulation model that would have the same dynamics of the avian-inspired airfoil section actuated by a SMA wire under a wind tunnel test. To achieve this goal, a deep neural network was implemented using initial wind tunnel data to learn to approximate the hardware dynamics. Figure 5.6 shows the learned model compared to the truth model in terms of temperature and displacement.



(a)



(b)

Figure 5.6: Preliminary results modeling the simple spring system showing (a) Displacement and temperature changes over time; and (b) Displacement as a function of temperature. Reprinted from [6].

5.4 Numerical Results

5.4.1 Validation of the Learning Algorithm

Before implementing the DDPG learning algorithm on the morphing wing experiment, it was the implementation was tested on a simulated inverted pendulum upswing. In this system, the algorithm must apply continuous positive or negative torque (u) to the pendulum shaft to keep it in the upright position. The only states available to the learning algorithm is the current angular position and velocity (θ and $\dot{\theta}$) of the pendulum. A reward signal was given to algorithm to represent its performance, as shown in Eq. (5.2), for every step. Reward results of more than -200 per episode are considered satisfactory because they numerically represent that the pendulum is held in vertical equilibrium after the swing. Figure 5.7a shows the resultant reward values per episode. It can be seen that the algorithm achieved the desired behavior of obtaining reward results of more than -200 on the final learning episodes, converging to a constant result. This can be interpreted as the learning algorithm, based on measured position and velocity and torque applied to the pendulum, being able to maximize the reward signal returned by the environment and complete the maneuver of an upswing in within 10 seconds.

To validate that the algorithm with same hyperparameters is able to achieve satisfactory behavior even with changes in the model, the same agent was trained on an inverted pendulum double the original mass. The algorithm has no knowledge that the environment was changed. Figure 5.7b shows the resultant reward values per episode for the double mass case. It can be observed that the same algorithm and neural network were able to learn a policy robust to changes in the environments and still approach the defined mark of -200 reward points that represent a complete inverted pendulum upswing maneuver. As an additional evaluation, Fig. 5.7c and 5.7d show the resultant reward values per episode for a double length pendulum case and for the half gravity case, respectively.

$$r = -(||\theta^2|| + .1(\dot{\theta}^2) + .001(u^2)) \tag{5.2}$$

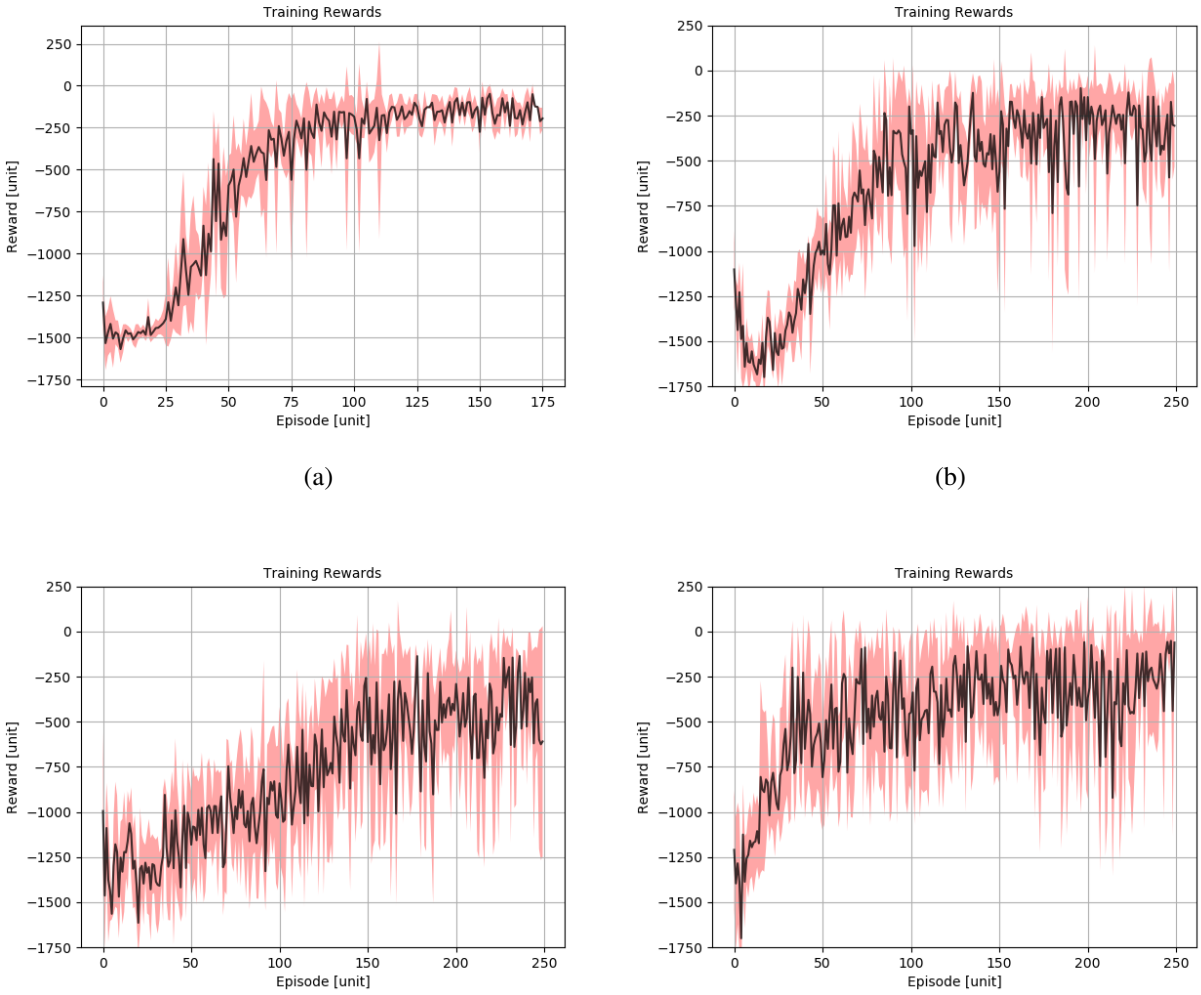
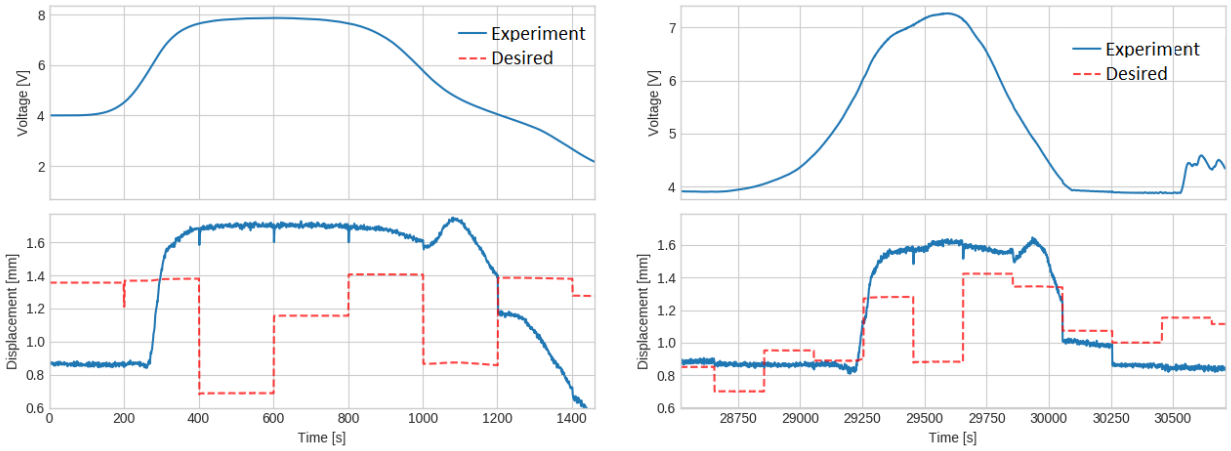


Figure 5.7: Results of the validation of the Deep Deterministic Policy Gradient algorithm solving the inverted pendulum upswing. Average of five runs and standard deviation of rewards per episode. (a) Unitary mass pendulum case; (b) Double mass pendulum case; (c) Double length pendulum case; and (d) Half gravity pendulum case. Reprinted from [6].

5.4.2 The Learning Algorithm in the Wind Tunnel

The main hypothesis is that the same algorithm used to learn to control different cases of the upswing pendulum task can be used to learn an shape change policy for morphing airfoils. The deep reinforcement learning algorithm was deployed to the hardware platform to test the hypothesis. The algorithm ran in the wind tunnel for more than 300 episodes. Every 200 time steps, or about

200 seconds, the commanded setpoint was randomly changed and the learning algorithm had to adapt the applied voltage to match the commanded value. It was intended to perform the training session for more than 300 episodes, but the wind tunnel operation was limited. Results achieved after 100 and 200 episodes are shown in Fig. 5.8. As expected, 300 episodes were not enough to train a policy that would satisfactory command the current displacement to the desired reference value, but it is possible to observe an improvement after 100 and 200 training episodes.



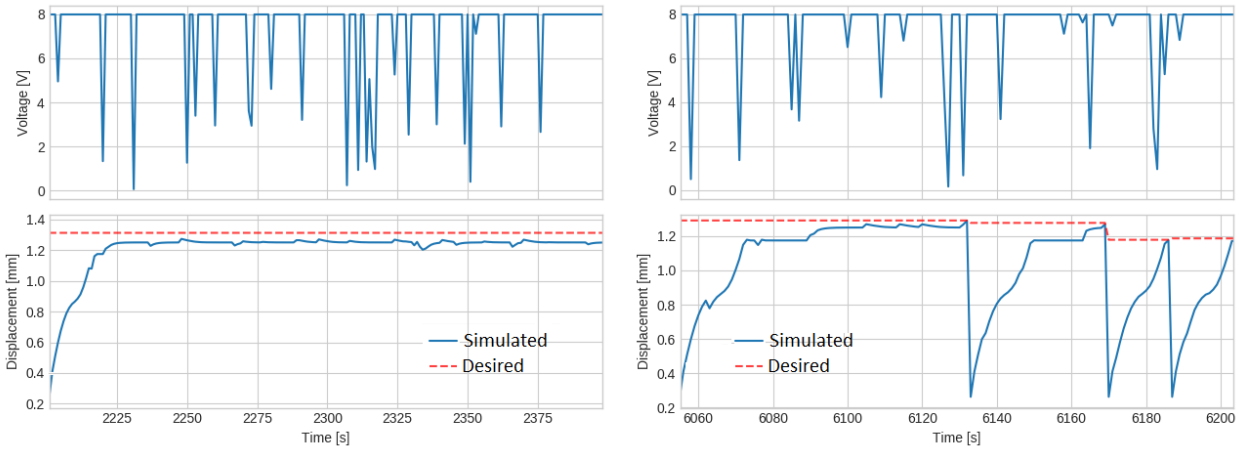
(a) (b)

Figure 5.8: Training on the Wind Tunnel: Deep Reinforcement Learning controller performance for training time consisting of: (a) 100 and (b) 200 episodes. Reprinted from [6].

5.4.3 The Learning Algorithm in the Simulation Model

As explained in Section 5.4.2, due to limited time at the wind tunnel, the learning algorithm was also tested using a simulated wing model created based on data collected from the wind tunnel. The algorithm ran for 1500 episodes. Every 200 time steps the commanded setpoint was randomly changed and the learning algorithm had to adapt the applied voltage to match the commanded value. Figure 5.9 shows the controller performance after 900 and 1500 episodes of training. After 900 episodes there is still steady-state error between the current and commanded displacement. After

1500 the deep reinforcement learning algorithm is able to complete the task multiple times in a row for different commanded morphing shapes.



(a) (b)

Figure 5.9: Training on the Modeled Airfoil: Deep Reinforcement Learning controller performance after 900 (a) and 1500 (b) episodes of training time, respectively. Reprinted from [6].

5.5 Summary

This paper addressed the challenge of controlling morphing air vehicles by developing a deep neural network and reinforcement learning technique for the control of shape memory alloy actuators that adapt the wing outer mold line. The objective was to investigate if the learning algorithm could also be used to learn satisfactory control policies for morphing wing. These are characterized by a plant which has continuous states and actions and time-varying dynamics. Based upon the results presented in the paper, the following conclusions are drawn.

1. Data collected from wind tunnel testing was shown to be effective in training a deep neural network that accurately mimics the dynamic behavior of the wing section actuated by a shape memory alloy. The same learning algorithm and hyper-parameters were also shown to control

shape changes from arbitrary initial shapes to arbitrary goal shapes when trained with the high-fidelity simulation model.

2. The deep reinforcement learning algorithm using the Deep Deterministic Policy Gradient learned to compute torques and perform an upswing maneuver on four different versions of an inverted pendulum. This algorithm was also shown to control a 3D-printed, avian-inspired airfoil section actuated by a shape-memory alloy operating in a wind tunnel. However, a fully satisfactory control policy was not learned due to time limitations on the testing platform.

3. The results provided insights that deep neural networks can be successfully used to model the dynamics of complex morphing air vehicle systems, followed by deep reinforcement learning algorithms to control these dynamic systems with continuous states and actions.

Further work on deep reinforcement learning algorithms are needed to reduce the extensive offline training required, and reduce the number of interactions with the environment. This would ease the transition to direct training on relevant hardware platforms.

6. CASE STUDY: INVERSE REINFORCEMENT LEARNING APPLIED TO GUIDANCE AND CONTROL OF SMALL UNMANNED AERIAL VEHICLES

This chapter was submitted under the name of "Inverse Reinforcement Learning Applied to Guidance and Control of Small Unmanned Aerial Vehicles" as final project of the 2018 Reinforcement Learning class at Texas A&M University, in Department of Electrical and Computer Engineering, taught by Dr. Dileep Kalathil, wrote by Vinicius G. Goecks, Akshay Sarvesh, Kishan P. Badrinath and is partially reproduced here.

In this chapter we investigate traditional Inverse Reinforcement Learning approaches and their extensions Guided Cost Learning and Generative Adversarial Imitation Learning to learn how to land a small Unmanned Aerial Vehicle based on human demonstration. This task is complex to traditional Inverse Reinforcement Learning algorithms due to the continuous nature of the observation and action-space. Our approach is first validated on less complex continuous tasks available on OpenAI Gym [209] — the Pendulum-v0 and LunarLanderContinuous-v2 environments — and later transferred to a high-fidelity UAV simulator, Microsoft AirSim [210]. Our results show that Generative Adversarial Imitation Learning is able to learn how to land an unmanned aerial vehicle and surpass human mean performance after 100 learning iterations. This approach learns from 100 human demonstrations of the landing task, equivalent to about 30 minutes of demonstration. The learning converges to the upper-bound of human performance after 400 learning iterations.

6.1 Problem Definition

It is true that humans can no longer claim to be the experts in tasks like image classification and recognition [211]. At the same time there are also a number of tasks where humans can easily outperform the machines, for now, in more complex tasks as, for example, driving and playing sports, mostly due to challenges in perception, motion control, and mechanical actuation.

A traditional Reinforcement Learning (RL) problem focuses on an agent learning a process

of decisions (policies) to produce a output which maximizes a reward function return by the environment. In Inverse Reinforcement Learning (IRL), the goal is to extract a reward functions by observing the behavior of an policy or any set of demonstrations. In this project we investigate traditional IRL approaches and their extensions Guided Cost Learning (GCL) and Generative Adversarial Imitation Learning (GAIL) to learn how to land a small Unmanned Aerial Vehicle (UAV) based on human demonstration. This task is complex for traditional IRL algorithms due to the continuous nature of the observation and action-space. Our approach is first validated on less complex continuous tasks available on OpenAI Gym [209] — the Pendulum-v0 and LunarLanderContinuous-v2 environments — and later transferred to a high-fidelity UAV simulator, Microsoft AirSim [210].

Section 6.2 of this chapter introduces the notation used throughout this document and the theoretical background of the algorithms used for this work. Section 6.3 details previous approaches for the IRL problem, highlighting their strengths and limitations. Section 6.4 details the application of IRL to learn the UAV landing task based on human demonstration, followed by results and discussion on Section 7.4. A summary is described in Section 7.5.

6.2 Notation and Background

6.2.1 Notation

A finite Markov Decision process (MDP) is a tuple $(S, A, \{P_{sa}\}, \gamma, R)$ where : S is a finite set of N $\mathbf{states}$, $A = \{a_1, a_2, .., a_k\}$ $\mathbf{actions}$, P_{sa} are the *transition probabilities* for taking an action a in state s, $\gamma \ \epsilon \ (0, 1]$ is the *discount factor*, R is the *reinforcement function* which is bounded by R_{max}.

A *policy* π is defined as a map from $\pi : S \rightarrow A$ and the value fn evaluated for any policy π is given by $V^{\pi}(s) = E[R(s_1) + \gamma R(S_2)+] + \gamma^2 R(s_3) + ... + \gamma^k R(s_k)]$. The Q fn can also be defined as : $Q^{\pi}(s, a) = R(s) + \gamma E[V^{\pi}(s)]$. The *optimal value fn* is $V^*(s) = sup_{\pi} V_{\pi}(s)$ and the *optimal Q fn is* $Q^*(s, a) = sup_{\pi} Q^{\pi}(s, a)$.

In a standard RL problem we find a policy π such that $V^{pi}(s)$ is maximized. In the IRL problem,

the goal is to find the best reward function for a set of observations.

6.2.2 Guided Cost Learning

Guided Cost Learning (GCL) [212] introduces an iterative sample-based method for estimating the probability normalization term Z in the Maximum entropy IRL formulation [34], and can scale to high-dimensional state and action spaces and non-linear reward structures. The algorithm estimates Z by training a new sampling distribution $q(\tau)$, with estimates of the expert trajectories as $\tilde{p}(\tau)$, general reward function parameterized by θ, and using importance sampling:

$$\mathcal{L}_{reward}(\theta) \quad = \quad \mathrm{E}_{\tau\sim p}[-r_\theta(\tau)] + \log\left(\mathrm{E}_{\tau\sim q}[\frac{\exp(r_\theta(\tau))}{\frac{1}{2}\tilde{p}(\tau)+\frac{1}{2}q(\tau)}]\right).$$

GCL alternates between optimizing r_θ using this estimate, and optimizing $q(\tau)$ to minimize the variance of the importance sampling estimate.

The optimal importance sampling distribution for estimating the partition function Z is proportional to the exponential family of the reward function. During GCL, the sampling policy $q(\tau)$ is updated to match this distribution by minimizing the Kullback-Leibler (KL or D_{KL}) divergence between $q(\tau$ and $\frac{1}{Z}\exp(r_\theta(\tau))$, or equivalently maximizing the learned reward and entropy:

$$\mathcal{L}_{sampler}(q) = \mathrm{E}_{\tau\sim q}[-r_\theta(\tau)] + \mathrm{E}_{\tau\sim q}[\log q(\tau)].$$

6.2.3 Generative Adversarial Imitation Learning

Generative Adversarial Imitation Learning (GAIL) [213] follows the generative modeling literature, in particular, Generative Adversarial Networks (GANs). Here, two models, a generator G and a discriminator D, are trained simultaneously. The discriminator is tasked with classifying its inputs as either the output of the generator, or actual samples coming from an expert's data distribution $p(\tau)$; where τ is sample trajectories for GAIL. The goal of the generator is to produce outputs that are classified by the discriminator as coming from the underlying data distribution.

To elucidate, the generator is subsumed by the environment generating trajectories ($\tau \sim G$)

from a random policy, while the discriminator takes as input a sample τ and outputs the probability $D(\tau)$ that the sample was from the data distribution. Since D and G are playing the same game but with opposing goals, their loss functions can be written as the following respectively:

$$\mathcal{L}(D) = \mathrm{E}_{\tau\sim p}[-\log D(\tau)] \quad + \quad \mathrm{E}_{\tau\sim G}[-\log(1 - D(\tau))]$$
$$\mathcal{L}(G) = \mathrm{E}_{\tau\sim G}[-\log D(\tau)] \quad + \quad \mathrm{E}_{\tau\sim G}[\log(1 - D(\tau))].$$

6.3 Related Work

Ng and Russell [32] defined the problem of IRL almost two decades ago. They state that the reward function is a more robust definition of a task rather than the policies themselves. The authors formulate the problem of IRL as a optimization problem and demonstrate algorithms to solve the Linear Programming (LP) problem on simple discrete, finite and continuous and infinite state problems.

Ziebart et al. [34] takes a different approach to matching feature counts which allow their formulation to avoid the ambiguity in getting many reward functions to derive policies; hence giving a unique, but randomized, solution as opposed to works like Abbeel and Ng [214]. They employ a tool, maximum entropy, from statistical mechanics developed by Jaynes [215], which prefers higher reward functions at an exponential rate in probability. So, maximizing the sum, meaning all possible observed trajectories, of logarithm of these probabilities across the reward function parameters, yields us with a stochastic solution for the reward function. But the suggested gradient descent algorithm has a drawback of recalculating the Markov Decision Process (MDP) in every step. So, we move on to the next work which incorporates these ideas and perform better.

Wulfmeier et al. [35] states that the objective of maximum entropy inverse reinforcement learning defined by Ziebart et al. [34], maximizing the joint posterior distribution of observing the expert demonstration under a given reward structure and model parameters, is fully differentiable with respect to deep neural network weights. Wulfmeier et al. [35] uses fully-connected convolutional networks (FCCN) to learn the reward model and evaluate the algorithm using *expected value*

difference: difference between value function for the optimal policy obtained using the learned reward model and using the ground truth reward. This approach achieves better results when compared to traditional maximum entropy IRL [34].

Finn et al. [212] extended MaxEntropy [34] formulation for solving high dimensional problems by making use of statistic's tool of importance sampling to estimate the partition function, which was the cause of hindrance in MaxEntropy problem. Ho and Ermon [213] presented a GAN-like algorithm for imitation learning, where the goal is to recover the policy that matched the export trajectories. Finn et al. [216] show that both GCL and GAIL are mathematically equivalent, that is, both converge to the same policy which imitates the expert policy. However, Ho and Ermon [213] use the typical unconstrained form of the discriminator and do not use the generator's density, and thus the reward function remains implicit within the discriminator and cannot be recovered.

6.4 Application to Guidance and Control of Small Unmanned Aerial Vehicles

More traditional IRL approaches as [32, 214, 34] requires the MDP to be solved at each learning iteration in order to optimize the policy for the learned reward function. In our UAV scenario, since the MDP is unknown, it is not possible to solve it at each iteration. Two recent algorithms, as explained in Section 6.2, proposes to address these issues: GCL and GAIL.

The GCL and GAIL algorithms are first validated on less complex continuous tasks available on OpenAI Gym [209] — the Pendulum-v0 and LunarLanderContinuous-v2 environments — and later transferred to a high-fidelity UAV simulator, Microsoft AirSim [210]. For the UAV scenario, the data was collected using the Microsoft AirSim environment [217] modified to simulate an small UAV perching task (landing) on top of a static vehicle, as seen in Figure 9.3. For this task, a UAV equipped with a bottom-facing RGB camera starts on a random location flying over the target vehicle and have to land in a certain amount of time. The states consists of inertial data of the UAV (estimated x, y, and z position, linear and angular velocities) and three additional features of the landing pad extracted using a custom computer vision module running on the background (radius, x and y pixel position of the center of the landing pad) — total of 15 features.

6.5 Results and Discussion

This section details the validation of the GCL and GAIL algorithms using the Pendulum-v0 and LunarLanderContinuous-v2 OpenAI Gym environments and the transition to Microsoft Airsim simulator to train an agent to perform the UAV landing task.

6.5.1 Numerical Results

For this work it was used an open-source implementation of GCL and GAIL algorithms [218] and started with simpler environments to evaluate and understand how the algorithms worked, what hyperparameters had the most impact in the performance, and how many expert demonstrations were necessary to learn a satisfactory policy.

This evaluation started by using the GCL algorithm on the OpenAI Gym Pendulum-v0 environment. The Pendulum-v0 task consists of applying a torque on the pendulum joint (one-dimensional continuous action) in order to keep it in a inverted vertical position. The observartion-space comprises three continuous features: $sin(\theta)$, $cos(\theta)$, and $\dot{\theta}$, where θ is the pendulum angle measured from the inverted vertical position, as seen in Figure 6.1.

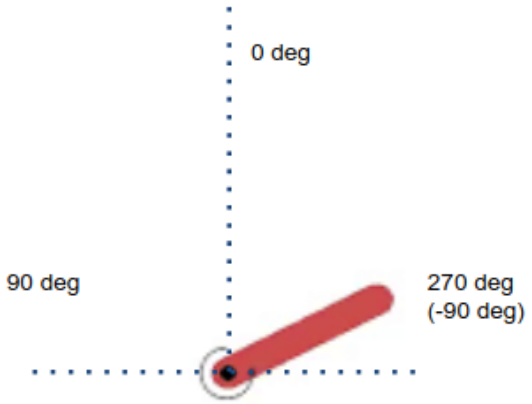


Figure 6.1: Illustration of the Pendulum environment. The agent controls the torque on the base of the pendulum in order to maintain it in the vertical position.

In order to evaluate the impact of the quantity and quality of the expert demonstrations to

initialize the GCL algorithm, we ran GCL on Pendulum-v0 initializing it with 5, 10, 15, and 20 close-to-optimal demonstrations and a final run with 200 demonstrations, in which half of them were close-to-optimal and the other half was sub-optimal. Results in terms of average return can be seen in Figure 6.2 when trained for 200 learning iterations. The algorithm performed the best with 10 close-to-optimal demonstration, showing that more demonstrations does not necessarily translates to better task performance.

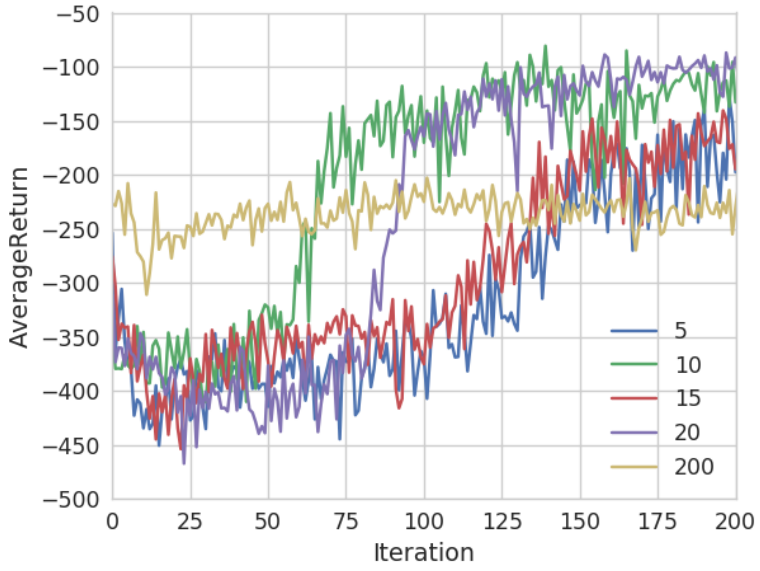


Figure 6.2: Performance in terms of original task reward for different number of expert trajectories for Pendulum GCL.

The GAIL algorithm was evaluated using the OpenAI Gym Lunar Lander Continuous environment. The LunarLanderContinuous-v2 task consists of controlling the main and side thrusts (two-dimensional continuous action) in order to safely land a spacecraft between to flags. The observartion-space comprises eight continuous and discrete features related to the spacecraft's position and attitude and contact to the ground, as illustrated in Figure 6.3.

In order to evaluate the impact of the quantity and quality of the expert demonstrations to initialize the GAIL algorithm, we ran GAIL on LunarLanderContinuous-v2 initializing it with 25,

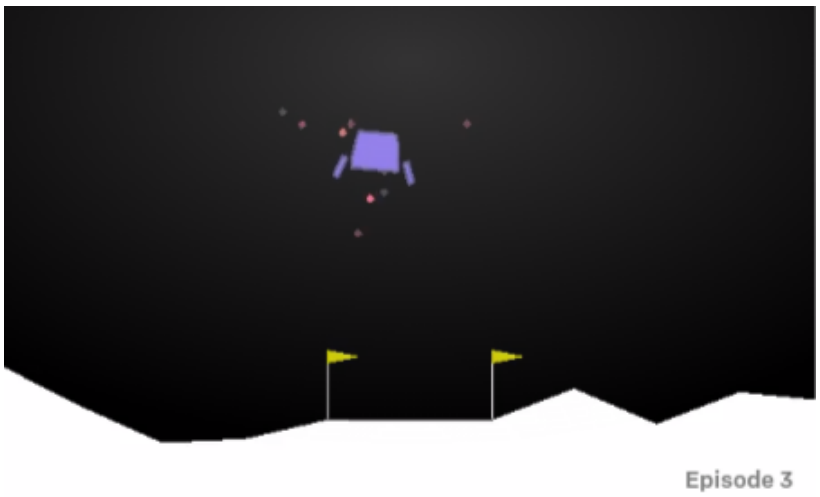


Figure 6.3: Illustration of the LunarLanderContinuous environment. The agent controls the main and side thrusts of the vehicle in order to land safely between he flags.

125, 1,250, 2,505, and 10,035 close-to-optimal demonstrations. Results in terms of average return can be seen in Figure 6.4 when trained for 500 learning iterations. The algorithm performed the best with 25 close-to-optimal demonstration, showing one more time that more demonstrations does not necessarily translates to better task performance. Figure 6.5 shows the performance of this better performing hyperparameter when trained for 1,000 learning iterations. The complete list of hyperparameters can be see in Table 6.1.

6.5.2 Unmanned Aerial Vehicle Results

After validation of the GAIL algorithm on a similar but less complex task, the Lunar Lander Continuous, we applied GAIL to solve the UAV landing task in Microsoft AirSim. A human pilot performed the landing task for approximately 30 minutes (or 100 demonstrations). This data was saved and used to initialize the GAIL algorithm. Results in terms of original task average return during the hyperparameter tuning phase can be seen in Figure 6.6 when trained for 500 learning iterations. Figure 6.7 details the performance of the better performing hyperparameter when trained for 500 learning iterations. For the UAV case, GAIL was able to surpass human mean performance after about 100 training iterations and converged to the human performance upper-bound after

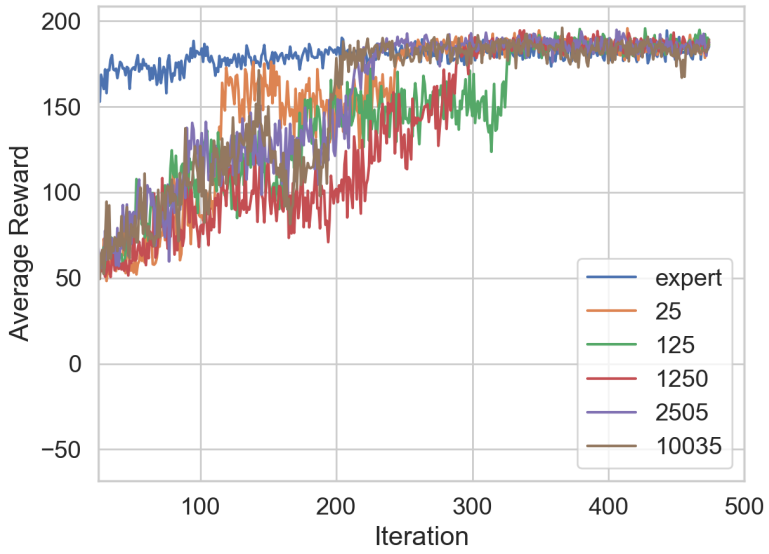


Figure 6.4: Performance in terms of original task reward for different number of expert trajectories for LunarLanderContinuous GAIL.

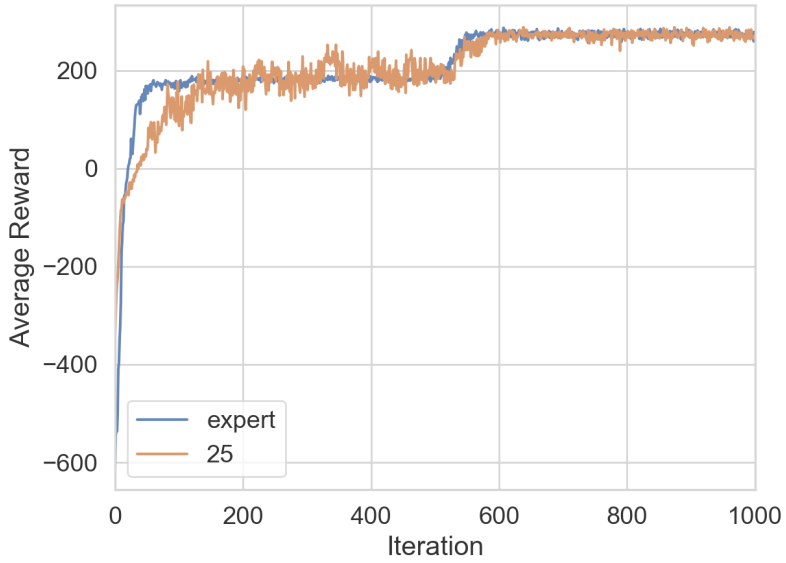


Figure 6.5: Performance in terms of original task reward for the best performing number of expert trajectories for LunarLanderContinuous GAIL.

about 400 learning iterations. We also compared GAIL to a pure RL approach using TRPO (Trust Region Policy Optimization), as also seen in Figure 6.7. TRPO only reached human performance by training iteration 500. The complete list of hyperparameters can be see in Table 6.2 and training statistics in Table 6.3.

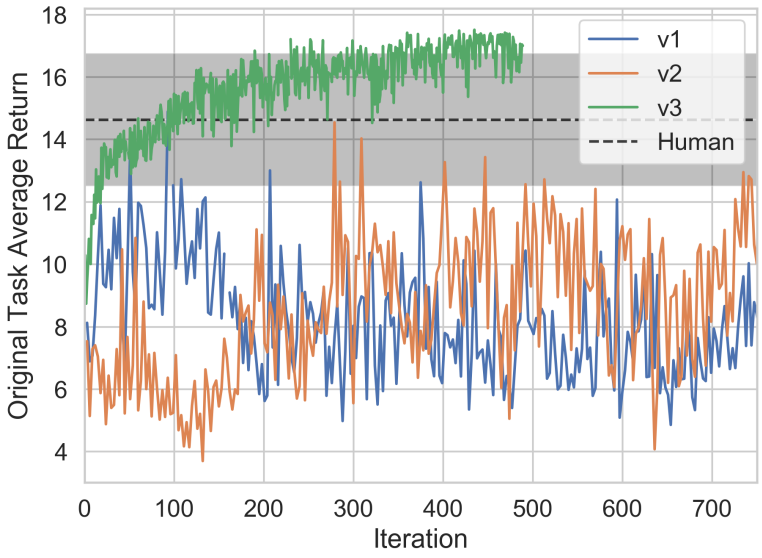


Figure 6.6: Performance in terms of original task reward for different hyperparameters for AirSim GAIL.

6.6 Summary

This chapter presented the application of IRL, specifically GCL and GAIL, to learn from previous task demonstration and solve continuous observation and action-space problems. Our approach using GCL and GAILwas initially validated in less complex environments, as the OpenAI Gym Pendulum-v0 and LunarLanderContinuous-v2, where we were able to solve the tasks with less than 200 learning iterations using as little as 10 demonstrations. The same GAIL approach was applied to a UAV landing task, where we were able to surpass human mean performance after 100 learning iterations. The same performance using pure RL, specifically TRPO, takes 400 learning

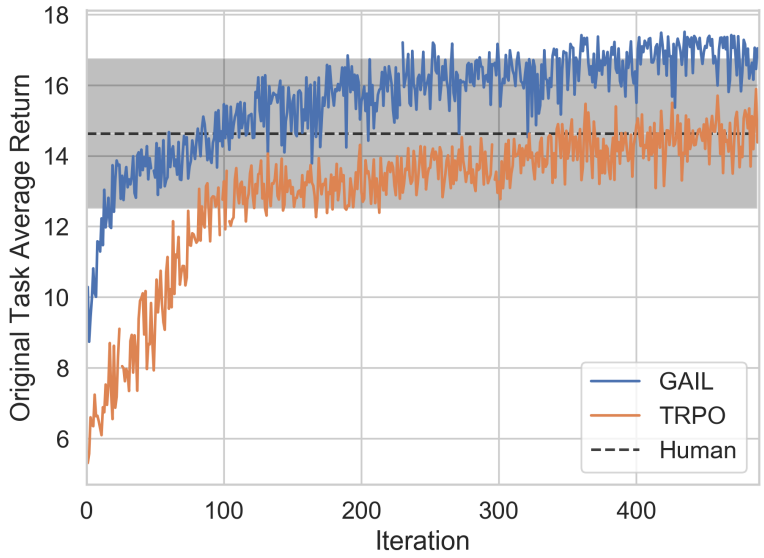


Figure 6.7: Performance in terms of original task reward for the best performing hyperparameters for AirSim GAIL.

iterations, or 400% more iterations. GAIL, as demonstrated for LunarLander (see Figure 6.4) and UAV (see Figures 6.6 and 6.7), is sample efficient in terms of number of expert trajectories needed it to reach a policy which is closer to the expert policy.

6.6.1 Training Hyperparameters

Table 6.1: Hyperparemeter search for LunarLanderContinuous GAIL

No. Expert Traj.	25	125	1250
Hyperparameter			
Policy NN	20, 20	30, 30	50, 50, 10
Batch Size	4000	7000	7000
Max Path Length	200	200	200
Discrim. Iters	100	100	100
Discount Factor	0.99	0.99	0.99 Policy NN
100, 100, 20	200, 200, 20	20, 20	
Batch Size	7000	7000	7000
Max Path Length	200	200	200
Discrim. Iters	100	100	0
Discount Factor	0.99	0.99	0.99

Table 6.2: Hyperparemeter search for AirSim GAIL

Hyperparameter	v1	v2	v3
Policy NN	50, 50, 10	32, 32	32, 32
Batch Size	60	60	1200
Max Path Length	60	60	60
Discrim. Iters	100	100	100
Discount Factor	0.99	0.99	0.99

Table 6.3: Training Statistics for AirSim GAIL

Training Time	48.12 hours
Trajectories Generated	10,429
GAIL Iterations	489
Timer per GAIL Iterations	$\sim$6 minutes
Time per Trajectory (average)	16.6 seconds
Human Trajectories	100
Human Time	28 minutes

7. CYBER-HUMAN APPROACH FOR LEARNING HUMAN INTENTION AND SHAPE ROBOTIC BEHAVIOR BASED ON TASK DEMONSTRATION*

Recent developments in artificial intelligence enabled training of autonomous robots without human supervision. Even without human supervision during training, current models have yet to be human-engineered and have neither guarantees to match human expectation nor perform within safety bounds. This paper proposes Cycle-of-Learning to leverage human-robot interaction and align goals between humans and robotic intelligent agents. Based on human demonstration of the task, Cycle-of-Learning learns an intrinsic reward function used by the human demonstrator to pursue the goal of the task. The learned intrinsic human function shapes the robotic behavior during training through deep reinforcement learning algorithms, removing the need for environment-dependent or hand-engineered reward signal. Two different hypotheses were tested, both using non-expert human operators for initial demonstration of a given task or desired behavior: one training a deep neural network to classify human-like behavior and other training a behavior cloning deep neural network to suggest actions. In this experiment, Cycle-of-Learning was tested in a high-fidelity unmanned air system simulation environment, Microsoft AirSim. The simulated aerial robot performed collision avoidance through a clustered forest environment using forward-looking depth sensing. The performance of Cycle-of-Learning is compared to behavior cloning algorithms and reinforcement learning algorithms guided by handcrafted reward functions. Results show that the human-learned intrinsic reward function can shape the behavior of robotic systems and have better task performance guiding reinforcement learning algorithms compared to standard human-handcrafted reward functions.

*Adapted with permission from "Cyber-human approach for learning human intention and shape robotic behavior based on task demonstration", by Vinicius G. Goecks, Gregory M. Gremillion, Hannah C. Lehman, and William D. Nothwang, presented at the 2018 International Joint Conference on Neural Networks (IJCNN) [7], Copyright 2018 by the Institute of Electrical and Electronics Engineers.

7.1 Problem Definition

Intelligent robots have the potential to positively impact and augment human activities in various tasks and modalities. Part of this success will depend on how they are integrated and the underlying motivation that drives the behavior of these intelligent robots. To comply with this need, it is beneficial to design a framework to train and shape the behavior of intelligent robotic agents to comply with human intention.

One of the approaches to shape the behavior of robotic agents is to formulate the process as a reinforcement learning problem. The robot (called *agent*) senses its surrounding environment (*observation*) through onboard sensors. The robot's controller (*policy*) selects that adequate control input (*action*) in order to maximize a given metric of performance (*reward signal*). The policy is trained based on its interaction with the environment in order to maximize the reward signal, which can be thought of as the goal of the intelligent agent.

This paper proposes the CyberSteer framework to better understand how to use human resources to train robotic agents in an environment without an explicit metric of performance (*reward signal*). The main research problem addressed by this paper is how to leverage initial human demonstration of the task to learn an intrinsic reward function used by the human to pursue the goal of the task. This learned intrinsic human function then shapes the robotic behavior during training through deep reinforcement learning (Deep RL) algorithms, aligning goals between humans and these intelligent agents.

Forming a reward signal is a challenge in real-world tasks. While games have clearly defined rewards from the game score, rewards in real-worlds tasks are currently handcrafted and hard-coded for each task based on some a priori knowledge of the possible state and clear goal. The challenge is in developing a method to translate human intention to a reward function in a framework that can be applied to arbitrary tasks.

CyberSteer provides a novel framework to integrate humans to robotic learning agents providing task demonstration and operating as an intervention mechanism to provide safety during learning and exploration. CyberSteer acts as an Intrinsic Reward Module (IRM) which provides the reward

signal for the Deep RL agent.

Two different hypotheses were tested, both using non-expert human operators for initial demonstration of a given task or desired behavior. CyberSteer #1 collected these demonstrated trajectories and trained a deep neural network to classify human-like behavior. CyberSteer #2 trained a behavior cloning deep neural network that asynchronously ran in the background suggesting actions to the Deep RL module.

The human is not required to be an expert on the task, only to have limited knowledge of its high-level goal and be able to perform rudimentary elements of the task.

The framework is designed for real-world robotic applications where external rewards are not present and safe environment exploration is a concern. CyberSteer is tested in a high-fidelity unmanned air system (UAS) simulation environment, Microsoft AirSim. The simulated aerial robot performs collision avoidance through a clustered forest environment using forward-looking depth sensing and roll, pitch, and yaw references angle commands to the flight controller.

The proposed approach is compared to a direct behavior cloning deep neural network trained using the human demonstration dataset and a deep reinforcement learning algorithm, Deep Deterministic Policy Gradient, guided by a handcrafted reward signal. Evaluation is quantified by the alignment of the agents actions with human inputs and completion of the task.

7.2 Background

7.2.1 Preliminaries

In reinforcement learning problems it is desired to train an agent to learn the parameters θ of a policy π_θ, in order to map the environment's state vectors $\boldsymbol{s}$ (sampled from a distribution $\mathcal{S}$) to agent actions $\boldsymbol{a}$ (sampled from a distribution $\mathcal{A}$). The performance of the agent is measured by a reward signal returned by the environment (external rewards, r_e) and/or returned by the agent itself (intrinsic rewards, r_i). At each time step t the reward signal can be computed as the sum of all the extrinsic and intrinsic rewards received (Equation 7.1).

$$r_t = r_{e_t} + r_{i_t} \tag{7.1}$$

An episode is defined as the time interval between the initial time step t_0 and the maximum number of time steps allowed T or until a previously established objective is achieved. The total reward per episode R is defined as the sum of the rewards received for each time step, as shown in Equation 7.2.

$$R = \sum_{t=0}^{T} r_t = \sum_{t=0}^{T} (r_{e_t} + r_{i_t}) \tag{7.2}$$

The expected total reward per episode received by a policy $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ can be defined by Equation 7.3:

$$R_{\pi_\theta} = \sum_{t=0}^{T} \mathop{\mathbb{E}}_{\boldsymbol{a}_t \sim \pi_\theta} [r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)] \tag{7.3}$$

7.2.2 Behavior Cloning

Behavior Cloning, or *Imitation Learning*, is defined by training a policy π in order to replicate an expert's behavior given states and actions visited during an expert demonstration

$$\mathcal{D} = \{\boldsymbol{a}_0, \boldsymbol{s}_0, \boldsymbol{a}_1, \boldsymbol{s}_1, ..., \boldsymbol{a}_T, \boldsymbol{s}_T\}\,.$$

This demonstration can be performed by a human supervisor, optimal controller, or virtually any other pre-trained policy π.

In the case of human demonstrations, the human expert is implicitly trying to maximize the reward function of a given task, as shown in Equation 9.1, where $\pi^*(\boldsymbol{a}_t^*|\boldsymbol{s}_t)$ represents the optimal policy (not necessarily known) in which the optimal action $\boldsymbol{a}^*$ is taken at state $\boldsymbol{s}$ for every time step t.

$$\max_{\boldsymbol{a}_0,\ldots,\boldsymbol{a}_T} \sum_{t=0}^{T} r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t=0}^{T} \log p(\pi^*(\boldsymbol{a}_t^*|\boldsymbol{s}_t)) \tag{7.4}$$

Defining the policy of the expert supervisor as π_{sup} and its estimated policy as $\hat{\pi}_{sup}$, behavior cloning can be achieved through standard supervised learning (where the parameters θ of a policy π_θ are fit in order to minimize a loss function - such as mean squared error - as shown in Equation 9.2) or using more advanced methods, such as Dataset Aggregation [219] (DAgger, where data collected by the estimated policy is aggregated with data provided by the expert), Generative Adversarial Imitation Learning [213] (where generative adversarial networks are used to fit distributions of states and actions defined by expert behavior), or Guided Cost Learning [212] (where regularized neural networks and a cost learning algorithm based on policy optimization are used to learn the expert's cost function under unknown dynamics and high-dimensional continuous systems).

$$\hat{\pi}_{sup} = \underset{\pi_\theta}{\operatorname{argmin}} \sum_{t=0}^{T} ||\pi_\theta(\boldsymbol{s}_t) - \boldsymbol{a}_t||^2 \tag{7.5}$$

7.2.3 Deep Deterministic Policy Gradient

The *Deep Deterministic Policy Gradient* (DDPG) algorithm [87] is a model-free, off-policy, actor-critic algorithm that combines the *Deterministic Policy Gradient* (DPG) algorithm [86] to compute the policy's gradient and the insights gained by the Deep Q-Network (DQN) algorithm [75, 94] to train the critic using Q-updates as, for example, the *Experience Replay* and *Target Networks*. Experience replay and target networks has been proven [75, 94, 220] to improve and stabilize policies learned using reinforcement learning. Both actor and critic are represented by deep neural networks.

The DDPG algorithm was selected for being able to process high-dimensional continuous state space, characteristic of robotic environments, and command continuous actions, similar to the human supervisor. To improve the state exploration performance in continuous spaces a noise component, sampled from a noisy process $\mathcal{N}$, is added to the agent's policy which affects the selected action.

7.3 Learning Human Intention and Shaping Robotic Behavior

7.3.1 The CyberSteer Mechanics

A summarized diagram of the mechanics of CyberSteer can be seen in Figure 7.1, which will support the explanation below.

Given an autonomous robotic agent initialized with an untrained policy (which maps the current observations to the next action), the human supervisor performs the desired task in order to collect demonstration data (observations and actions used by the human) safely performing an initial environment exploration. From this point, two hypotheses are being tested to efficiently use this human data:

1. **CyberSteer #1** (Detailed in Algorithm 2) Trains a convolutional recurrent neural network based on collected human data and random behavior to classify actions taken as being provided by human or not (human-like actions), Figure 7.2. The estimated reward $\hat{r}$ to be

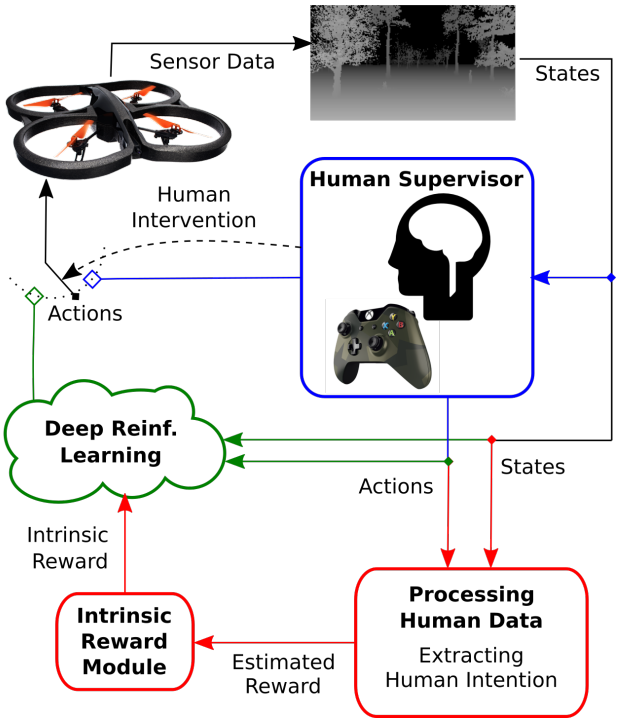


Figure 7.1: Overall Diagram of the CyberSteer framework. Reprinted from [7].

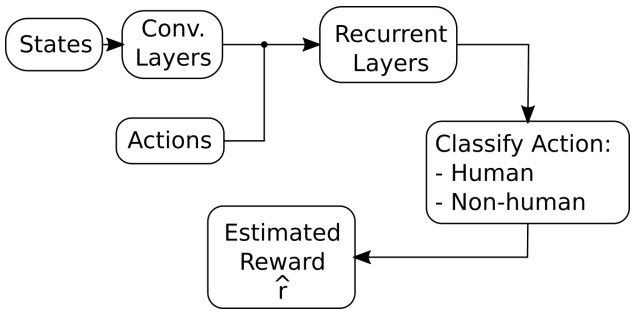


Figure 7.2: CyberSteer #1: Computing estimated rewards based on the likelihood of the action taken being similar to previous actions taken by the human supervisor. Reprinted from [7].

returned to the Deep RL algorithm is calculated based on the likelihood that the action taken is similar to the action executed by a human $p(\boldsymbol{a} = \boldsymbol{a}_{human}|(\boldsymbol{s}, \boldsymbol{a}))$, as shown in Equation 7.6.

$$\hat{r} = r_{max}(2 \cdot p(\boldsymbol{a} = \boldsymbol{a}_{human}|(\boldsymbol{s}, \boldsymbol{a})) - 1) \tag{7.6}$$

The estimated reward is bounded between r_{max} (when $p(\boldsymbol{a} = \boldsymbol{a}_{human}|(\boldsymbol{s}, \boldsymbol{a})) = 1$) and $-r_{max}$ (when $p(\boldsymbol{a} = \boldsymbol{a}_{human}|(\boldsymbol{s}, \boldsymbol{a})) = 0$). CyberSteer # 1 uses a convolutional network with two convolutional layers (32 and 16 filters, 3x3 kernels, 1 pixel stride, no padding, using rectified linear unit (ReLU) activation function and max pooling to reduced the size of the input in half after each layer), 8 LSTM recurrent units, two fully-connected layers with 64 neurons each, followed by a ReLU activation function, and the fully connected output layer, followed by a sigmoid function. *Adam* algorithm [207] is used for optimization of a binary cross-entropy loss function.

2. **CyberSteer #2** (Detailed in Algorithm 3) Trains a behavior cloning neural network using convolutional and recurrent layers to suggest actions to the Deep RL algorithm that would have been taken by the human supervisor, Figure 7.3. The mean square error ($\boldsymbol{a}_{MSE}$) between the action suggested by the behavior cloning ($\boldsymbol{a}_{BC}$) and taken by the Deep RL algorithm ($\boldsymbol{a}_{DRL}$) is used to compute an estimated reward (Equation 7.7, where $\boldsymbol{a}_{MSE} = (\boldsymbol{a}_{BC} - \boldsymbol{a}_{DRL})^2$) to guide the Deep RL algorithm.

Algorithm 2 CyberSteer #1

1: Human controls the robot and demonstrates the task to be executed. States ($\boldsymbol{s}$) and actions ($\boldsymbol{a}$) are recorded using onboard sensors and compiled as a human-controlled dataset $\mathcal{D}_H = \{\boldsymbol{a}_0, \boldsymbol{s}_0, \boldsymbol{a}_1, \boldsymbol{s}_1, ..., \boldsymbol{a}_T, \boldsymbol{s}_T\}$.
2: Robot performed magnitude-controlled random actions. States ($\boldsymbol{s}$) and actions ($\boldsymbol{a}$) are recorded using onboard sensors and compiled as a non-human-controlled dataset $\mathcal{D}_{NH}$.
3: Datasets $\mathcal{D}_H$ and $\mathcal{D}_{NH}$ are reorganized in order to have three sequences of actions stacked as one data point.
4: Datasets $\mathcal{D}_H$ and $\mathcal{D}_{NH}$ are aggregated and shuffled.
5: Convolutional recurrent neural network is trained using the aggregated dataset in order to classify between human and non-human behavior - the CyberSteer #1 network.
6: Controls are completely transferred to robot, and its Deep Reinforcement Learning (Deep RL) module is initialized. Robot is trained:
7: **while** Desired behavior is not achieved **do**
8: Robot's Deep RL module selects next action.
9: **if** Human judges robot's behavior is inadequate **then**
10: Human intervention. Provides new demonstration.
11: Batch update CyberSteer #1 network.
12: **else**
13: CyberSteer #1 network evaluates from 0 to 100% how similar the action is to human behavior.
14: Estimated reward is computed based on Equation 7.6 by the Intrinsic Reward Module (IRM).
15: Update Robot's Deep RL module.

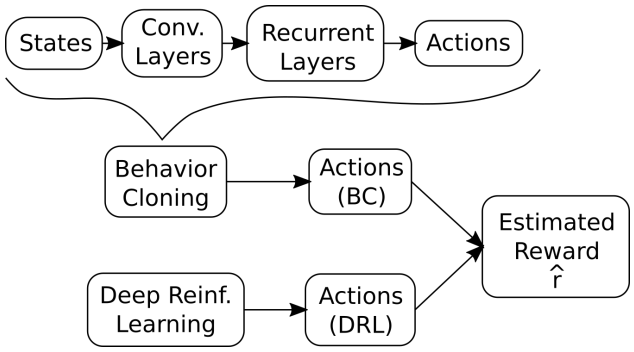


Figure 7.3: CyberSteer #2: Computing estimated rewards based on how similar the actions taken by the agent are when compared to a behavior cloning network trained with demonstration form the human supervisor. Reprinted from [7].

$$\hat{r} = -r_{max}\,(1 - \sqrt{\boldsymbol{a}_{MSE}}) \tag{7.7}$$

The estimated reward is bounded between r_{max} (when $\boldsymbol{a}_{MSE} = 0$) and $-r_{max}$ (when $\boldsymbol{a}_{MSE} = 4$). CyberSteer # 2 uses a convolutional network with two convolutional layers (32 and 16 filters, 3x3 kernels, 1 pixel stride, no padding, using rectified linear unit (ReLU) activation function and max pooling to reduce the size of the input in half after each layer), 8 LSTM recurrent units, one fully-connected layers with 64 neurons followed by a ReLU activation function, and the fully connected output layer followed by a sigmoid function. *Adam* algorithm [207] is used for optimization of a binary cross-entropy loss function.

7.3.2 Environment and Task Modeling

7.3.2.1 Unmanned Air System Simulation Environment

The proposed approach was first developed and tested using the Microsoft AirSim platform [210] - a high-fidelity simulated environment for unmanned air systems (UAS) flight and control. This platform simulates the UAS flight dynamics, onboard inertial sensors (GPS, barometer, gyroscope, accelerometer, and magnetometer), onboard camera data (monocular RGB, ground truth depth sensor, and random image segmentation), and collision and environment physics.

The authors refer the reader to the original paper [210] for more details about the architecture,

Algorithm 3 CyberSteer #2

1: Human controls the robot and demonstrates the task to be executed. States ($\boldsymbol{s}$) and actions ($\boldsymbol{a}$) are recorded using onboard sensors and compiled as a human-controlled dataset $\mathcal{D}_H = \{\boldsymbol{a}_0, \boldsymbol{s}_0, \boldsymbol{a}_1, \boldsymbol{s}_1, ..., \boldsymbol{a}_T, \boldsymbol{s}_T\}$.
2: Dataset $\mathcal{D}_H$ is reorganized in order to have three sequences of actions stacked as one data point and dataset is shuffled.
3: Behavior Cloning network is trained is order to suggest actions based on past human demonstration.
4: Controls are completely transferred to robot, and its Deep Reinforcement Learning (Deep RL) module is initialized. Robot is trained:
5: **while** Desired behavior is not achieved **do**
6: Robot's Deep RL module selects next action.
7: **if** Human judges robot's behavior is inadequate **then**
8: Human intervention. Provides new demonstration.
9: Batch update CyberSteer #2 network (Behavior Cloning).
10: **else**
11: CyberSteer #2 network (Behavior Cloning) suggests action based on current robot's states.
12: Estimated reward is computed based on Equation 7.7 by the Intrinsic Reward Module (IRM).
13: Update Robot's Deep RL module.

simulation parameters, accuracy of the model, and flight characteristics when compared to a real UAS flying in real-world.

7.3.2.2 Collision Avoidance Scenario

The proposed algorithm was first tested on a collision avoidance scenario where the agent had to maneuver the unmanned air system using roll commands while automatic flying forward at constant altitude. A warehouse environment was created using Unreal Engine for visually realistic textures and objects. A sample of the forest scenario used can be seen in Figure 7.4.

7.4 Numerical Results

Building onto the behavior cloning concepts explained in Subsection 7.2.2, it was desired to validated that the autonomous agent would be able to learn, to some degree, to perform the demonstrated collision avoidance maneuvers using the behavior cloning network. The behavior cloning network is a critical component of the proposed framework as it is subsequently used to

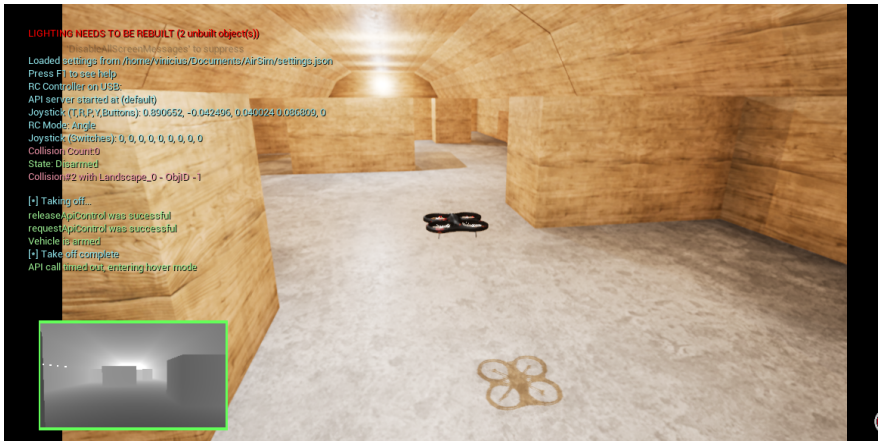


Figure 7.4: Warehouse scenario created for the collision avoidance task using Unreal Engine for visually realistic textures and objects. Reprinted from [7].

evaluate the actions proposed by the Deep RL algorithm for one of the hypotheses.

A non-expert human piloted the unmanned air system performing collision avoidance maneuvers for 12,000 time steps (approximately 20 minutes, performing actions at 20 Hz). The UAS was set to have constant altitude and constant forward velocity and images were collected from the onboard depth sensor and resized to 36 by 64 pixels. The human sent roll angle control inputs to steer laterally and avoid incoming obstacles. These images and control inputs were compiled as human-controlled dataset $\mathcal{D}_H$.

A convolutional network with three convolutional layers (32, 64, and 128 filters, 3x3 kernels, 1 pixel stride, no padding, using rectified linear unit (ReLU) activation function and max pooling to reduced the size of the input in half after each layer) and two fully-connected layers connected by a 50% dropout layer during training was trained to imitate human performance. Three frames were stacked together to extract velocity of the scene. For the first experiment, the network was trained offline and deployed to the agent for testing. Video of the learning performance can be seen at: https://www.youtube.com/watch?v=rEbNytWz3c8.

The behavior cloning experiment was followed by the evaluation of both hypotheses. The main goal was to create an Intrinsic Reward Module (IRM) that would be able to guide a Deep RL algorithm.

CyberSteer # 1 combined the human-controlled dataset $\mathcal{D}_H$ (collected for the behavior cloning network) to a dataset of images and control inputs generated at random by the robot performing the task: the non-human-controlled dataset $\mathcal{D}_{NH}$. While the agent was performing random actions, the human had total control authority to intervene and control the robot to prevent any damage to the platform. Both datasets, $\mathcal{D}_H$ and $\mathcal{D}_{NH}$, were aggregated and shuffled and the CyberSteer # 1 network was trained to classify image-action pairs between human and non-human actions associating a number between 0 and 1 based on how similar the image-action pairs were compared to the collected human behavior. CyberSteer # 1 network was deployed to the IRM in order to guide a Deep RL algorithm (Deep Deterministic Policy Gradient: DDPG) by using this similarity metric as a reward function.

CyberSteer # 2 uses the collected human-controlled dataset $\mathcal{D}_H$ to train a behavior cloning network. When the Deep RL agent is controlling the robot, CyberSteer # 2 IRM runs asynchronously in the background comparing the actions taken by the agent and what would have been taken by the CyberSteer # 2 network (behavior cloning). Based on how similar the action taken is when compared to the suggested one, a reward signal is generated and guides the Deep RL algorithm.

A human reward baseline was established based on the maximum possible total reward for the task. In the case of CyberSteer #1, all actions proposed by the Deep RL would be classified with 100% confidence by the IRM as being human-like actions. In the case of CyberSteer #2, all actions proposed by the Deep RL would 100% match the actions suggested by the IRM running a behavior cloning network based on the initial demonstration of the task. As a baseline, the proposed solution was compared to the same Deep RL algorithm being fed by a standard human-engineered reward function returned by the environment based on the task completion, i.e., the fraction of the environment's total distance traversed.

Figure 7.5 shows the result of this first comparison for 500 episodes of the task being performed for about 30 seconds. CyberSteer #2 performed seven orders of magnitude better than CyberSteer #1 and the baseline.

Figure 7.6 shows the comparison in terms of task completion using the proposed solution and the

Figure 7.5: Comparison of the proposed solutions to achieve human-like performance on the task with no feedback from the environment when compared to a baseline dependent on environment reward signals. Reprinted from [7].

established baseline. Even though motivated by different reward signals both proposed approaches and the baseline performed similar in terms of task completion during training.

Since we are using deep neural networks as learning representation, there is still no theoretical convergence guarantee that the algorithm will lead to the optimal task behavior.

7.5 Summary

This paper proposed CyberSteer to leverage human-robot interaction to train autonomous robots aligning their goals to comply with human intention. CyberSteer frames the training task as a reinforcement learning problem: an agent interacting with the environment to maximize a reward signal that represents the goal of the task. The learned human intention is instantiated as an intrinsic reward signal to shape the robotic behavior during training when combined with Deep Reinforcement Learning algorithms.

Two different hypotheses were tested to translate human intention as reward signal, both using non-expert human operators for initial demonstration of a given task or desired behavior. CyberSteer

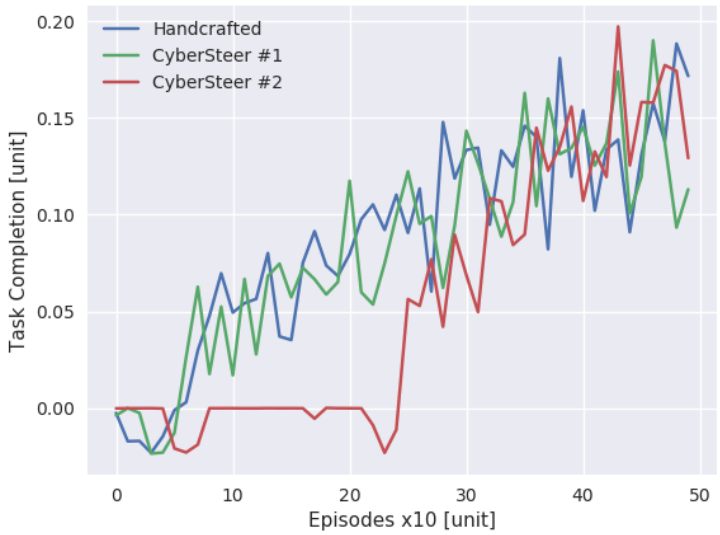


Figure 7.6: Task completion performance of the proposed solutions with no feedback from the environment when compared to the established baseline dependent on environment reward signals. Reprinted from [7].

#1 collected these demonstrated trajectories and trained a deep neural network to classify human-like behavior. CyberSteer #2 trained a behavior cloning deep neural network that asynchronously ran in the background suggesting actions to the Deep Reinforcement Learning module.

In this experiment, CyberSteer was tested in a high-fidelity unmanned air system simulation environment, Microsoft AirSim. The simulated aerial robot performed collision avoidance through a clustered forest environment using forward-looking depth sensing.

CyberSteer #2 initially showed seven-fold improvement in performance when compared to a human-engineered approach when training autonomous agents. The approach showed that modern Deep Reinforcement Learning algorithms can be adapted to be guided by reward signals reconstructed from human demonstration, with no need to have a human expert to handcraft a reward function. This enables the same algorithm to be deployed in different real-world applications, as long as the task can be demonstrated beforehand.

Both CyberSteer approaches were still able to perform similarly well to standard approaches dependent on environment-returned and human-engineered reward signals when driven by modern

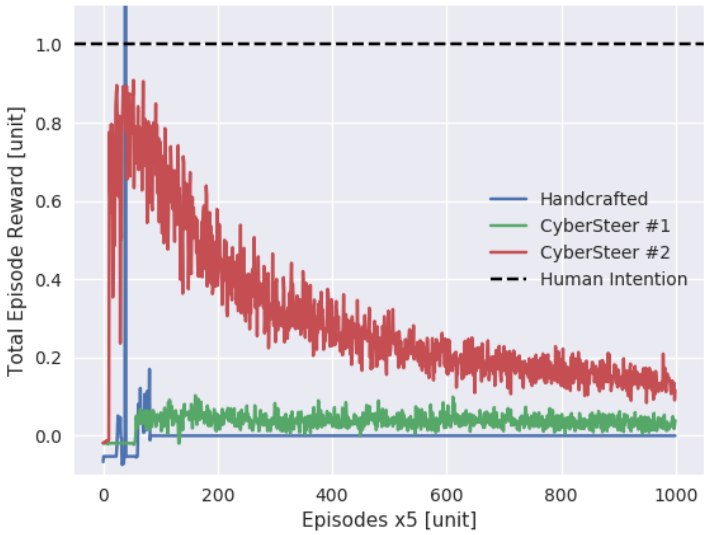


Figure 7.7: Extended plots of the comparison of the proposed solutions to achieve human-like performance on the task with no feedback from the environment when compared to a baseline dependent on environment reward signals. Reprinted from [7].

Deep Reinforcement Learning algorithms.

Currently, both approaches show decrease of performance after the first 100 episodes, as seen in Figures 7.7 and 7.8. This will be further investigated when performing different tasks.

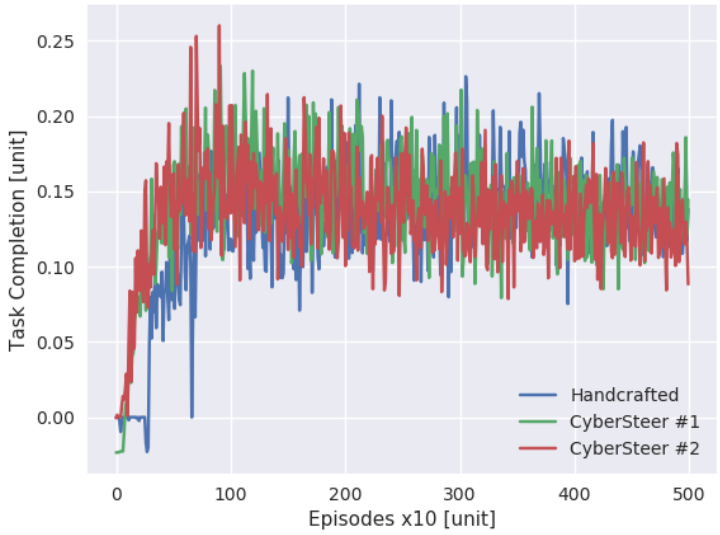


Figure 7.8: Extended plots of the task completion performance of the proposed solutions with no feedback from the environment when compared to the established baseline dependent on environment reward signals. Reprinted from [7].

8. CYCLE-OF-LEARNING FOR AUTONOMOUS SYSTEMS FROM HUMAN INTERACTION*

We discuss different types of human-robot interaction paradigms in the context of training end-to-end reinforcement learning algorithms. We provide a taxonomy to categorize the types of human interaction and present our Cycle-of-Learning framework for autonomous systems that combines different human-interaction modalities with reinforcement learning. Two key concepts provided by our Cycle-of-Learning framework are how it handles the integration of the different human-interaction modalities (demonstration, intervention, and evaluation) and how to define the switching criteria between them.

8.1 Problem Definition

Reinforcement learning (RL) has been successfully applied to solve challenging problems from playing video games to robotics. In simple scenarios, a reward function or model of the environment is typically available to the RL algorithm, and standard RL techniques can be applied. In real-world physical environments where reward functions are not available or are too intractable to design by hand, these standard RL methods often tend to fail. Using humans to train robotic systems is a natural way to overcome these burdens.

There have been many examples in the field of human-robot interaction where human interaction is used to train autonomous systems in the context of end-to-end reinforcement learning. For example, imitation learning is an approach where demonstrations of a task provided from a human are used to initially train an autonomous system to learn a policy that "imitates" the actions of the human. A recent approach termed Human Centered Reinforcement learning (HCRL) trains autonomous systems using positive and negative feedback provided from a human trainer, showing promising strides for learning policies in the absence of a reward function. Approaches that learn

*Adapted with permission from "Cycle-of-Learning for Autonomous Systems from Human Interaction", by Nicholas R. Waytowich, Vinicius G. Goecks, Vernon J. Lawhern, presented at the 2018 Artificial Intelligence and Human-Interaction (AI-HRI) AAAI Fall Series Symposium [1], available under a Creative Commons license at the AI-HRI/2018/05 proceedings at https://arxiv.org/html/1809.06606.

control policies jointly with humans and autonomous systems acting together in a shared autonomy framework have also been developed.

Each of these human-interaction approaches, which have their own unique advantages and disadvantages, have mostly been utilized in isolation without taking into consideration the varying degrees of human involvement; the human is a valuable, but finite, source of information that can communicate information in many ways. A key question to consider is not only *what* information the human should convey, but *how* this information should be conveyed. A method for combining these different paradigms is needed to enable efficient learning from human interaction.

In this paper, we present a new conceptual framework for using humans to train autonomous systems called the Cycle-of-Learning for Autonomous Systems from Human Interaction. This framework fuses together different human-interaction modalities into a single learning paradigm inspired by how humans teach other humans new tasks. We believe this intuitive concept should be employed whenever humans are interacting with autonomous systems. Our contributions in this paper are as follows: we first describe a taxonomy of learning from human interaction (with corresponding literature) and list the advantages and disadvantages of each learning modality. We then describe the Cycle-of-Learning (Figure 1.1), our conceptual framework for intuitively combining these interaction modalities for efficient learning for autonomous systems. We describe the different stages of our framework in the context of the degree of human involvement (and, conversely, the amount of autonomy of the system). We conclude with potential avenues for future research.

8.2 Types of Learning from Human Interaction

There are many different ways a human may be used to train an autonomous system to perform a task, each of which can be broadly categorized by the modality of human-interaction. Here we present a taxonomy that categorizes most types of human interactions into one of three methodologies: Learning from human demonstrations (LFD), learning from human interventions (LFI) and learning from human evaluations (LFE). This taxonomy is partitioned based on the amount of control that the human or Autonomous system has during the learning process. In LFD the

human is providing demonstrations and is in full control. In LFI, where the human occasionally intervenes, both the human and autonomous system share control. In LFE, where the human is providing evaluative feedback as the autonomous system performs a task, the autonomous systems is in control. We describe rec ent research efforts for each of these categories and present their relative advantages and disadvantages.

8.2.1 Learning from Human Demonstrations

Learning from human Demonstrations (LFD) is from a broad class of techniques called Imitation Learning (IL) where the aim is to train autonomous systems to mimic human behavior in a given task. In this interaction paradigm, the human takes the role of a demonstrator to provide examples of the task in the terms of sequences of states and actions. Using these demonstrations, the autonomous system learns a policy (a mapping from states to actions) that mimics the human demonstrations.

There are many empirical successes of using imitation learning to train autonomous systems. For self-driving cars, Bojarski et al. successfully used IL to train a policy that mapped from front-facing camera images to steering wheel commands using around one hundred hours of human driving data [19]. A similar approach was taken to train a small unmanned air system (sUAS) to navigate through forest trails, but in this case data was collected by a human following the trail [20].

These results highlight both advantages and disadvantages of imitation learning approaches. IL can be used end-to-end to learn a new task, overcoming initial exploration of randomly-initialized learning algorithms while having better convergence properties because it trains on static datasets. However, the major drawback is that policy performance and generalization rely on the diversity and quality and size of the training dataset — often requiring large amounts of demonstrated behavior.

Inverse Reinforcement Learning (IRL), also known as apprenticeship learning, is another form of learning from human demonstrations where a reward function is learned based on task demonstration. The idea is that during the demonstration, the human is using a policy that is optimal with respect to some internal reward function. However, in IRL, the quality of the reward function learned is still dependent on the demonstration performance. In addition, IRL is fundamentally underdefined (degenerated), in a sense that different reward functions can lead to the same behavior [32, 212].

IRL focuses on learning the reward function, which is believed to be a more robust and transferable definition of the task when compared to the policy learned [32].

8.2.2 Learning from Human Interventions

Learning From human Interventions (LFI) is a much less explored method of human interaction for training autonomous systems. Simply put, the human acts as an overseer and intervenes (i.e. takes control) when the autonomous system is about to enter a catastrophic state. This is especially important for training embodied autonomous systems, such as quadrotors or ground robots, that are learning and interacting in a real environment where sub-optimal actions could lead to damage to the systems itself or the surrounding environment. Recently, this concept was formalized in a framework called learning from Human Interventions for safe Reinforcement Learning (HIRL) [39]. In HIRL, when the human observes the agent perform sub-optimal actions that could lead to a failure state, the human "intervenes" by blocking the current action and providing an alternative action. Additionally, a negative reward signal is given to the AI system during the intervention to learn from.

By using humans to perform an initial state exploration and prevent catastrophic actions, LFI is a promising approach to solve the exploitation-exploration dilemma and increase safety in RL. Additionally, there is evidence that shared autonomy scenarios can also improve human performance [45]. However, LFI by itself does not scale to more complex environments where millions of observations are required for successful learning, increasing the required amount of human supervision [39]).

8.2.3 Learning from Human Evaluations

Another human-interaction modality that we consider is Learning From human Evaluations (LFE). In LFE, the human acts as a supervisor and provides real-time evaluations (or critiques) to interactively shape the behavior of the autonomous system. There are several different approaches for how to provide the human feedback for LFE. One of the simplest approaches is fitting a reward function based on binarized feedback; for example, "good" vs "bad" actions, indicating positive and

negative reward, respectively. Existing frameworks that use this approach include TAMER [46, 48] and COACH [47]. Another approach asks the human to rank-order a given set of *trajectories*, or short movie clips representing a sequence of state-action pairs, and learning a reward function in a relational manner based on human preferences [56]. Both approaches have been shown to be effective across a variety of domains and illustrates the utility of using human-centered reward shaping for shaping the policy of autonomous systems.

An advantage of LFE techniques is that they do not require the human to be able to perform demonstrations and only require an understanding of the task goal. However, if the time-scale of the autonomous system is faster than human reaction time, then it can be challenging for the autonomous system to attribute which actions correspond to the provided feedback. In addition, the human reward signals are generally non-stationary and policy-dependent, i.e.: what was a good action in the past may not be a good action in the present depending on the humans perception of the autonomous system's policy.

8.3 The Cycle-of-Learning Concept

The Cycle-of-Learning is a framework for training autonomous systems through human interaction that is based on the intuition on how a human would teach another human to perform a new task. For example, teachers conveying new concepts to their students proceed first by demonstrating the concept, intervening as needed while students are learning the concept, then providing critique after students have started to gain mastery of the concept. This process is repeated as new concepts are introduced. While extensive research has been conducted into each of these stages separately in the context of machine learning and robotics, to the best of our knowledge, a model incorporating each of these aspects into one learning framework has yet to be proposed. We believe such a framework will be important to fielding adaptable autonomous systems that can be trained on-the-fly to perform new behaviors depending on the task at hand, in a manner that does not require expert programming.

Under the proposed Cycle-of-Learning framework (Figure 1.1), we start with LFD where a human would be asked to provide several demonstrations of the task. This demonstration data (observations received and actions taken) constitute the initial human dataset D_H. The dataset

Algorithm 4 Cycle-of-Learning Framework

1: **procedure** LEARNING FROM DEMONSTRATION
2: **while** $SwitchingFunction_{LFD}$: **do**
3: Collect human demonstration data D_H
4: Train imitation learning policy $\pi_{\theta D_H}$
5: Learn human reward function R_H from D_H
6: **procedure** LEARNING FROM INTERVENTION
7: **while** $SwitchingFunction_{LFI}$: **do**
8: Autonomous system performs the task
9: **if** Human intervenes **then**:
10: Collect human intervention data D_I
11: Aggregate $D_H \leftarrow D_H \cup D_I$
12: Update imitation learning policy $\pi_{\theta D_H}$
13: Update human reward function R_H
14: Compute intervention reward R_I
15: Train critic Q_H using R_I and TD error
16: Update policy $\pi_{\theta D_H}$ using policy gradient
17: **procedure** LEARNING FROM EVALUATION
18: **while** $SwitchingFunction_{LFE}$: **do**
19: Collect human evaluation reward r_H
20: Update critic Q_H using r_H and TD error
21: Update policy $\pi_{\theta D_H}$ using policy gradient
22: Update human reward function R_H with r_H
23: **procedure** REINFORCEMENT LEARNING
24: **while** $SwitchingFunction_{RL}$: **do**
25: Autonomous system performs the task
26: Compute rewards using R_H
27: Update critic Q_H using R_H and TD error
28: Update policy $\pi_{\theta D_H}$ using policy gradient

D_H feeds an imitation learning algorithm to be trained via supervised learning, resulting in the policy $\pi_{\theta D_H}$. In parallel to the policy training, the dataset D_H is used by an Inverse Reinforcement Learning (IRL) algorithm to infer the reward function R_H used by the human while demonstrating the task (Algorithm 4, line 1).

On LFI (Algorithm 4, line 6) the autonomous system performs the task according to the policy $\pi_{\theta D_H}$. During the task the human is able to intervene by taking over the control of the autonomous system, perhaps to avoid catastrophic failure, and provides more demonstrations during

the intervention. This new intervention dataset D_I is aggregated to the previous human dataset D_H. Using this augmented dataset, the policy $\pi_{\theta D_H}$ and the reward model R_H are updated. An intervention reward R_I is computed based on the degree of the intervention. The reward signal R_I and the temporal-difference (TD) error associated with it are used to train a value function Q_H (the critic) and evaluate the actions taken by the actor. At this point, the policy $\pi_{\theta D_H}$ is updated using actor-critic policy gradient methods.

After the human demonstration and intervention stages, the human assumes the role of a supervisor who evaluates the autonomous system actions through a reward signal r_H — Learning from Evaluation (LFE, Algorithm 4, line 17). Similarly to the LFI stage, the reward signal r_I and the TD error associated with it are used to update the critic Q_H and the policy $\pi_{\theta D_H}$. The reward model R_H is also updated according to the signal r_H plus the observations and actions associated with it.

The final stage is pure Reinforcement Learning (RL). The autonomous system performs the task and its performance is evaluated using the learned reward model R_H (Algorithm 4, line 23). Similar to the LFI and LFE stages, the reward signal R_H and the TD error associated with it are used to update the critic Q_H and the policy $\pi_{\theta D_H}$. This sequential process is repeated as new tasks are introduced.

8.3.1 Integrating and Switching Between Human Interaction Modalities

Two key concepts of the Cycle-of-Learning framework are how to handle the integration of the learned models from the different interaction modalities (demonstration, intervention, and evaluation) and how to define the criteria to switch between them. First, to integrate the different interaction modalities, we propose using an actor-critic architecture [221]: initially training only the actor, and later adding the critic. Training the actor first allows the framework to leverage the initial demonstration and intervention provided by the human. The critic is then trained as the human assumes the role of supervisor. After enough human demonstration data has been collected we can infer a reward function through IRL. At the end, the actor and critic are combined on a standard actor-critic reinforcement learning architecture driven by the learned reward model.

Second, we propose different concepts to define a criteria to switch between interaction modalities: performance metrics, data modality limitation, and advantage functions. *Performance metrics:* A pre-defined performance metric can be used to indicate when to switch modalities once the policy reaches a certain level. Alternatively, the human interacting with the system could manually switch between different interaction modalities as s/he observes that the autonomous system performance is not increasing. *Data modality limitations:* Depending on the task, there can be a limited amount of demonstration, intervention, or evaluation that can be provided by humans. In this case, the framework switches between modalities according to data availability. *Advantage functions:* After training the reward model R_H, advantages $A(s, a)$ (the difference between the state-action value function $Q(s, a)$ and the state value function $V(s)$, which compares the expected return of a given state-action pair to the expected return on that state) can be computed and used for expected return comparison between human and autonomous systems actions. With this information, the framework could switch interaction modalities whenever the advantage function of the autonomous system surpasses the advantage function of the human. These, as well as other potential concepts for modality switching, need to be further investigated and can be adapted to meet task requirements.

8.4 Summary

This paper presents the Cycle-of-Learning framework, envisioning the integration between different human-interaction modalities and reinforcement learning algorithms in an efficient manner. The main contributions of this work are (1) the formalization of the underlying learning architecture — first leveraging human demonstrations and interventions to train an actor policy and reward model, then gradually moving to training a critic and fine-tuning the reward model based on the same interventions and additional evaluations, to finally combining these different parts on an actor-critic architecture driven by the learned reward model and optimized by a reinforcement learning algorithm — and (2) the switching between these human-interaction modalities based on performance metrics, data modality limitations, and/or advantage functions.

We believe the proposed Cycle-of-Learning framework is most suitable for robotic applications, where both human and autonomous system resources are valuable and finite. As future work, it is

planned to demonstrate these techniques on a human-sUAS (small unmanned air system) scenario.

9. EFFICIENTLY COMBINING HUMAN DEMONSTRATIONS AND INTERVENTIONS FOR SAFE TRAINING OF AUTONOMOUS SYSTEMS IN REAL-TIME*

This paper investigates how to utilize different forms of human interaction to safely train autonomous systems in real-time by learning from both human demonstrations and interventions. We implement two components of the Cycle-of-Learning for Autonomous Systems, which is our framework for combining multiple modalities of human interaction. The current effort employs human demonstrations to teach a desired behavior via imitation learning, then leverages intervention data to correct for undesired behaviors produced by the imitation learner to teach novel tasks to an autonomous agent safely, after only minutes of training. We demonstrate this method in an autonomous perching task using a quadrotor with continuous roll, pitch, yaw, and throttle commands and imagery captured from a downward-facing camera in a high-fidelity simulated environment. Our method improves task completion performance for the same amount of human interaction when compared to learning from demonstrations alone, while also requiring on average 32% less data to achieve that performance. This provides evidence that combining multiple modes of human interaction can increase both the training speed and overall performance of policies for autonomous systems.

9.1 Problem Definition

The primary goal of learning methodologies is to imbue intelligent agents with the capability to autonomously and successfully perform complex tasks, when *a priori* design of the necessary behaviors is intractable. Most tasks of interest, especially those with real-world applicability, quickly exceed the capability of designers to handcraft optimal or even successful policies. It can even be infeasible to construct appropriate objective or reward functions in many cases. Instead, learning techniques can be used to empirically discover the underlying objective function for the

*Adapted with permission from "Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real time", by Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich, presented at the 2019 AAAI Conference on Artificial Intelligence [8], Copyright 2019 by the Association for the Advancement of Artificial Intelligence.

task and the policy required to satisfy it, typically utilizing state, action, or reward data. Several classes of these techniques have yielded promising results, including learning from demonstration, learning from evaluation, and reinforcement learning.

Reinforcement learning has been proven to work on scenarios with well-designed reward functions and easily available interactions with the environment [76]. However, in real-world robotic applications, explicit reward functions are non-existent, and interactions with the hardware are expensive and susceptible to catastrophic failures. This motivates leveraging human interaction to supply this reward function and task knowledge, to reduce the amount of high-risk interactions with the environment, and to safely shape the behavior of robotic agents.

Learning from evaluation is one such way to leverage human domain knowledge and intent to shape agent behavior through sparse interactions in the form of evaluative feedback, possibly allowing for the approximation of a reward function [46, 47, 48]. This technique has the advantage of minimally tasking the human evaluator and can be used when training behaviors they themselves cannot perform. However, it can be slow to converge as the agent can only identify desired or even stable behaviors through more random exploration or indirect guidance from human negative reinforcement of unwanted actions, rather than through more explicit examples of desired behaviors.

In such a case, learning from demonstration can be used to provide a more directed path to these intended behaviors by utilizing examples of the humans performing the task. This technique has the advantage of quickly converging to more stable behaviors. However, given that it is typically performed offline, it does not provide a mechanism for corrective or preventative inputs when the learned behavior results in undesirable or catastrophic outcomes, potentially due to unseen states. Learning from demonstration also inherently requires the maximal burden on the human, requiring them to perform the task many times until the state space has been sufficiently explored, so as to generate a robust policy. Also, it necessarily fails when the human is incapable of performing the task successfully at all.

Learning from interventions, where a human acts as an overseer while an agent is performing a task and periodically takes over control or intervenes when necessary, can provide a method to

improve the agent policy while preventing or mitigating catastrophic behaviors [39]. This technique can also reduce the amount of direct interactions with the agent, when compared to learning from demonstration. Similar to learning from evaluation, this technique suffers from the disadvantage that desired behaviors must be discovered through more variable exploration, resulting in slower convergence and less stable behavior.

Most of these human interaction methods have been studied separately, and there is very little work combining multiple modalities to leverage strengths and mitigate weaknesses. In this paper, we work towards our conceptual framework that combines multiple human-agent interaction modalities into a single framework, called the Cycle-of-Learning for Autonomous Systems from Human Interaction [1]. Our goal is to unify different human-in-the-loop learning techniques in a single framework to overcome the drawbacks of training from different human interaction modalities in isolation, while also maintaining data-efficiency and safety.

In this paper, we present our initial work towards this goal with a method for combining learning from demonstrations and learning from interventions for safe and efficient training of autonomous systems. We seek to develop a real-time learning technique that combines demonstrations as well as interventions provided from a human to outperform traditional imitation learning techniques while maintaining agent safety and requiring less data. We validate our method with an aerial robotic perching task in a high-fidelity simulator using a quadrotor that has continuous roll, pitch, yaw and throttle commands and a downward facing camera. In particular, the contributions of our work are twofold:

1. We propose a method for efficiently and safely learning from human demonstrations and interventions in real-time.

2. We empirically investigate both the task performance and data efficiency associated with combining human demonstrations and interventions.

We show that policies trained with human demonstrations and human interventions together outperform policies trained with just human demonstrations while simultaneously using less data. To the

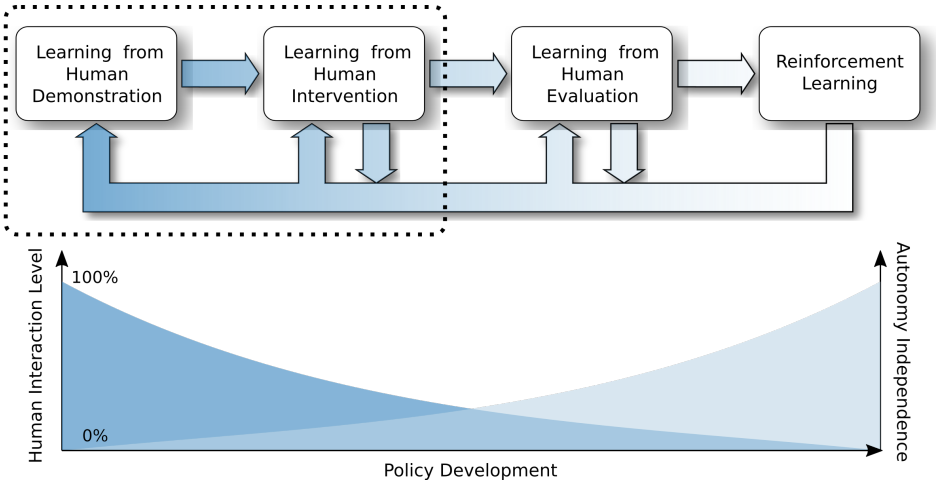


Figure 9.1: Cycle-of-Learning for Autonomous Systems from Human Interaction: a concept for combining multiple forms of human interaction with reinforcement learning. As the policy develops, the autonomy independence increases and the human interaction level decreases. This work focuses on the first two components of the cycle (dashed box): Learning from Demonstration and Learning from Intervention. Reprinted from [8].

best of our knowledge this is the first result showing that training a policy with a specific sequence of human interactions (demonstrations, then interventions) outperforms training a policy with just human demonstrations (controlling for the total amount of human interactions), and that one can obtain this performance with significantly reduced data requirements, providing initial evidence that the role of the human should adapt during the training of safe autonomous systems.

9.2 Background and Related Work

9.2.1 Learning from Demonstrations

Here we provide a brief summary of Learning from Demonstrations (LfD); a more comprehensive review can be found in [11]. Learning from Demonstrations, sometimes referred to as *Imitation Learning*, is defined by training a policy π in order to generalize over a subset $\mathcal{D}$ of states and actions visited during a task demonstration over T time steps:

$$\mathcal{D} = \{\boldsymbol{a}_0, \boldsymbol{s}_0, \boldsymbol{a}_1, \boldsymbol{s}_1, ..., \boldsymbol{a}_T, \boldsymbol{s}_T\} .$$

This demonstration can be performed by a human supervisor, optimal controller, or virtually any other pre-trained policy.

In the case of human demonstrations, the human is implicitly trying to maximize what may be represented as an internal reward function for a given task (Equation 9.1), where $\pi^*(\boldsymbol{a}_t^*|\boldsymbol{s}_t)$ represents the optimal policy that is not necessarily known, in which the optimal action $\boldsymbol{a}^*$ is taken at state $\boldsymbol{s}$ for every time step t.

$$\max_{\boldsymbol{a}_0,\ldots,\boldsymbol{a}_T} \sum_{t=0}^{T} r_t(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{t=0}^{T} \log p(\pi^*(\boldsymbol{a}_t^*|\boldsymbol{s}_t)) \tag{9.1}$$

Defining the policy of the supervisor as π_{sup} and its estimate as $\hat{\pi}_{sup}$, imitation learning can be achieved through standard supervised learning, where the parameters θ of a policy π_θ are trained in order to minimize a loss function, such as mean squared error, as shown in Equation 9.2.

$$\hat{\pi}_{sup} = \underset{\pi_\theta}{\operatorname{argmin}} \sum_{t=0}^{T} ||\pi_\theta(\boldsymbol{s}_t) - \boldsymbol{a}_t||^2 \tag{9.2}$$

There are many empirical successes of using imitation learning to train autonomous systems. For self-driving cars, Bojarski et al. successfully used human demonstrations to train a policy that mapped from front-facing camera images to steering wheel commands using around one hundred hours of human driving data [19]. Similar approaches have been taken to train small unmanned air system (sUAS) to navigate through cluttered environments while avoiding obstacles, where demonstration data was collected by human oracles in simulated [7] and real-world environments [20].

9.2.2 Learning from Interventions

In *Learning from Interventions* (LfI) the human takes the role of a supervisor and watches the agent performing the task and intervenes (i.e. overriding agent actions with human actions) when necessary, in order to avoid unsafe behaviors that may lead to catastrophic states. Recently, this learning from human intervention concept was used for safe reinforcement learning (RL) that could train model-free RL agents without a single catastrophe [39]. Similar work has proposed

using human interaction to train a classifier to detect unsafe states, which would then trigger the intervention by a safe policy previously trained based on human demonstration of the task [40]. This off-policy data generated by the safe policy is aggregated to the replay buffer of a value-based reinforcement learning algorithm (Double Deep Q-Network, or DDQN [80]). The main advantage of this method is being able to combine the off-policy data generated by the interventions to update the current policy.

9.2.3 Related Work

Several existing works have studied, in isolation, the use of different human interaction modalities to train policies for autonomous systems, either in the form of demonstrations [37], [11], interventions [38], [39] or evaluations [46]. However, there has been relatively little work on how to effectively combine multiple human interaction modalities into a single learning framework. Several cases include the combination of demonstrations and mixed initiative control for training robot polices [43] as well as the recent work by Hilleli and El-Yaniv where imitation learning was combined with interactive reward shaping in a simulated racing game [40] and the recent work [222] where deviation from the expert demonstration is added to a reward function to be optimized with reinforcement learning.

Another example of work that attempts to augment learning from demonstrations with additional human interaction is the Dataset Aggregation (DAgger) algorithm [26]. DAgger is an iterative algorithm that consists of two policies, a primary agent policy that is used for direct control of a system, and a reference policy that is used to generate additional labels to fine-tune the primary policy towards optimal behavior. Importantly, the reference policy's actions are not taken, but are instead aggregated and used as additional labels to re-train the primary policy for the next iteration. In [27] DAgger was used to train a collision avoidance policy for an autonomous quadrotor using imitation learning on a set of human demonstrations to learn the primary policy and using the human observer as a reference policy. There are some drawbacks to this approach that are worth discussing. As noted in [27], because the human observer is never in direct control of the policy, safety is not guaranteed, since the agent has the potential to visit previously unseen states, which could cause

catastrophic failures. Additionally, the subsequent labeling by the human can be suboptimal both in the amount of data recorded (perhaps recording more data in suboptimal states than is needed to learn an optimal policy) as well as in capturing the intended result of the human observer's action (as in distinguishing a minor course correction from a sharp turn, or the appropriate combination of actions to perform a behavior). Another limitation of DAgger is that the human feedback was provided *offline* after each run while viewing a slower replay of the video stream to improve the resulting label quality. This prevents the application to tasks where real-time interaction between humans and agents are required.

9.3 Combining Learning from Human Demonstrations and Interventions

This work demonstrates a technique for efficiently training an autonomous system safely and in real-time by combining learning from demonstrations and interventions. It is the first part of the Cycle-of-Learning concept (Figure 9.1) which aims to combine multiple forms of human-agent interaction for learning a policy that mimics the human trainer in a safe and efficient manner. Although this paper focuses on the first two parts of the Cycle-of-Learning, for brevity, we will refer to the algorithm presented here as the Cycle-of-Learning (CoL).

The CoL starts by training an initial policy π_0 from a set of task demonstrations provided by the human trainer using a standard supervised learning technique (regression in this case since the action-space for our task is continuous). Next, the agent is given control and executes π_0 while the human takes the role of overseer and supervises the agent's actions. Using a joystick controller, the human intervenes whenever the agent exhibits unwanted behavior that diverges from the policy of the human trainer, and provides corrective actions to drive the agent back on course, and then releases control back to the agent. The agent then learns from this intervention by augmenting the original training dataset with the states and actions from the intervention, and then fine-tuning π_0. The agent then executes the new policy π_n while the human continues to oversee and provides interventions as necessary. In practice, the human trainer can easily switch between providing demonstrations and interventions by switching control between the human and the agent as shown in Figure 9.2. Combining demonstration and intervention data in this way should not only improve

the policy over what learning from demonstration can do alone but also require less training data to do so. The intuition is that the agent will inevitably end up in states previously unexplored with the original demonstration data which will cause it's policy to fail and that intervening from those failure states allows the agent to quickly learn from those deficiencies or "blind spots" in its own policy in a more targeted fashion than from demonstration data alone [223]. In this way, we learn only from the critical states, which is more data efficient, instead of using all states for training as is done in DAgger [26].

9.3.1 Data Efficiency

A demonstration is defined as a human-produced trajectory of state-action pairs for the entire episode, while an intervention is defined as a trajectory of state-action pairs for only the subset of the episode where corrective action is deemed necessary by the human. Thus, the amount of data provided via intervention is nearly always less than the amount provided via demonstration. Training routines that incorporate more episodes utilizing learning from intervention rather than learning from demonstration will in general be more data sparse, assuming comparable task performance. Therefore, by utilizing components of the CoL to learn from both demonstration and intervention, we can train with less data than if demonstrations had been used in isolation for an equivalent number of episodes, resulting in a more efficient training framework. This concept generalizes to the full CoL, as the agent naturally requires less input from the human as its policy develops, its task proficiency increases, and it becomes more autonomous (indicated in Figure 9.1).

9.3.2 Safe Learning

The notion of *safe* learning here refers to the ability of a human oracle to intervene in cases where catastrophic failure may be imminent. Thus, the agent is able to explore higher risk regions of the state space with a greater degree of safety. This approach leverages human domain knowledge and ability to forecast such boundary states, which the agent cannot do early in the training process when the state space is less explored. By allowing the policy to explore less seen regions and then provide training data of how to correct from those states, human interventions provide a richer

dataset that improves the policy in those regimes. This is contrasted to a method based solely on demonstration, which may only see states and observations along a nominal trajectory and have a policy poorly fit to data outside that envelope. The result is a policy that is more robust, through greater data diversity, while not risking damage to the agent that is typical with methods that rely on random exploration of the state space. This provides a method to safely train an autonomous system.

9.3.3 Real-Time Interaction

The utility of the demonstrated approach is partially linked to the ability of the agent to consume data as it is provided by the human oracle and update its policy online. The current system accomplishes this by storing all subject state-action pairs in the training dataset, which is queried in real-time to update the policy, and then fine-tuning that policy whenever new samples are added to the dataset. During intervention, this allows for interaction with an agent using a policy trained on the most recent corrective actions provided by the human. The short time between novel human intervention data and behavioral roll-outs from the agent policy prevents significant delay in this feedback loop that might result from more infrequent, batch learning. As in closed loop systems, large temporal delays between feedback inputs and their resultant output behaviors can lead to instability. In this context, that would manifest as unstable training as the human oracle would need to correct for undesired actions for significantly longer before seeing any effect on the agent behavior. This shortcoming was exhibited in DAgger, where policy correction was a delayed, offline process.

9.4 Implementation

The next sections address the experimental methodology used to evaluate the proposed approach and the implementation of the learning algorithm (shown in Figures 9.2 and Algorithm 6).

9.4.1 Environment Modeling

We tested our CoL approach (Figure 9.2) in an autonomous quadrotor perching task using a high-fidelity drone simulator based on the Unreal Engine called AirSim developed by Microsoft

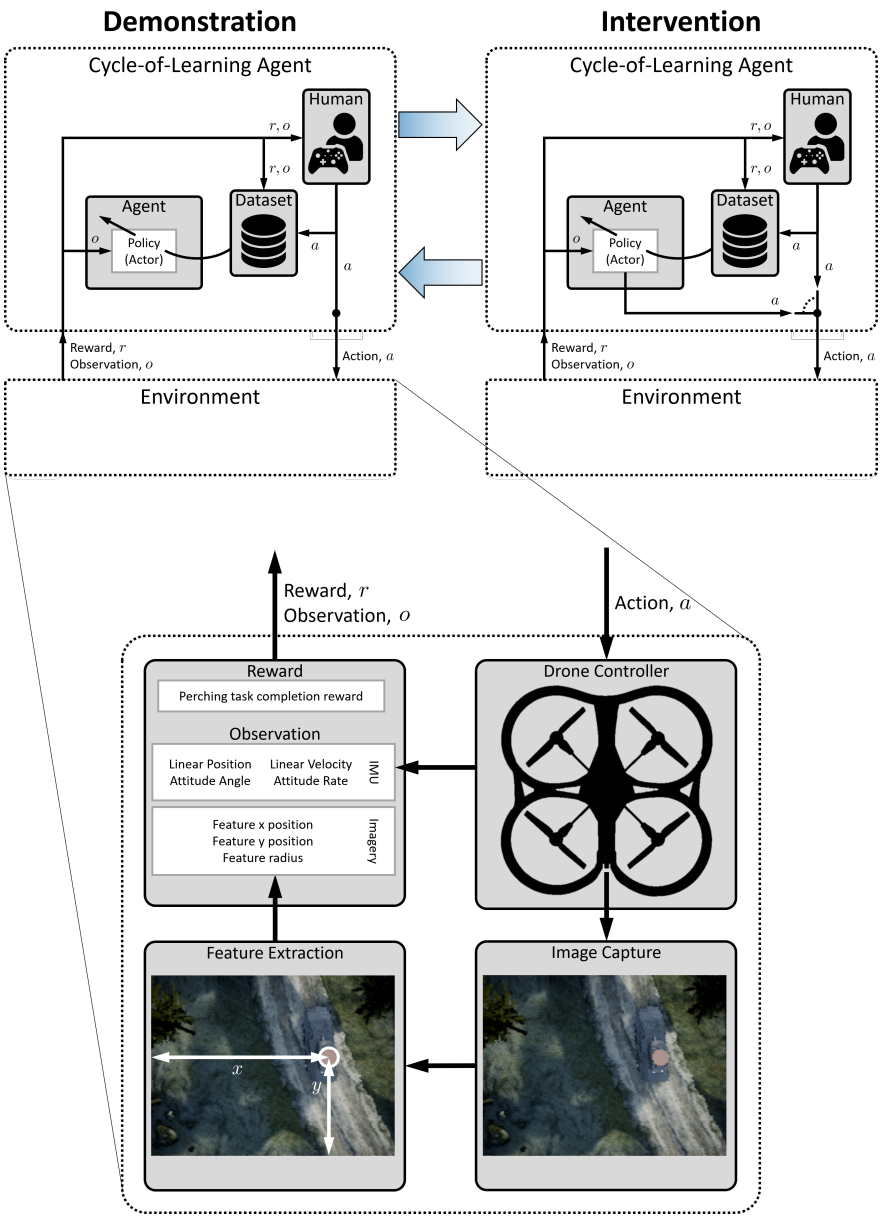


Figure 9.2: Flow diagram illustrating the learning from demonstration and intervention stages in the CoL for the quadrotor perching task. Reprinted from [8].

[217]. AirSim provides realistic emulation of quad-rotor vehicle dynamics while the Unreal Engine allows for the development of photo-realistic environments. In this paper, we are concerned with training a quadrotor to autonomously land on a small landing platform placed on top of a ground vehicle (see Figure 9.3).

The current observation-space consists of vehicle inertial and angular positions, linear and

Figure 9.3: Screenshot of AirSim environment and landing task. Inset image in lower right corner: downward-facing camera view used for extracting the position and radius of the landing pad which is part of the observation-space that the agent learns from. Reprinted from [8].

angular velocities, and pixel position and radius of the landing pad extracted using a hand-crafted computer vision module (15 dimensional continuous observation-space). The vehicle is equipped with a downward-facing RGB camera that captures 320×240 pixel resolution images. The camera framerate and agent action frequency is $10.5\,\text{Hz}$, while the human observer views the video stream at approximately $35\,\text{Hz}$. The action-space comprises the four continuous joystick commands (roll, pitch, yaw, and throttle), which are translated to reference velocity commands (lateral, longitudinal, heading rate, and heave) by the vehicle autopilot.

For the perching task, the goal is to land the quadrotor as close to the center of the landing pad as possible. We define a landing a success if the quadrotor lands within 0.5m radius of the center of the platform and a failure otherwise. At the beginning of each episode, the quadrotor starts in a random x,y location at a fixed height above the landing pad and the episode ends when either the quadrotor reaches the ground (or landing pad) or after 500 time-steps have elapsed.

9.4.2 The Cycle-of-Learning Algorithm

As shown in Algorithm 6 the main procedure starts by initializing the agent's policy π, the human dataset $\mathcal{D}_H$, the *Update Policy* subroutine, and task performance threshold. The main loop

Algorithm 5 Combining Human Demonstrations and Interventions (Cycle-of-Learning)

procedure MAIN
    Initialize agent's policy π
    Initialize human dataset $\mathcal{D}_H$
    Initialize *Update Policy* procedure
    Define performance threshold α
    while task performance $< \alpha$ **do**
        Read observation o
        Sample action $a_\pi \sim \pi$
        if Human Interaction (a_H) **then**:
            Perform human action a_H
            Add o and a_H to $\mathcal{D}_H$
        else
            Perform a_π
        if End of Episode **then**
            Evaluate task performance
procedure UPDATE POLICY
    Spawn separate thread
    Initialize $loss$ threshold $loss_{TH}$
    while *Main* procedure running **do**
        Load human dataset $\mathcal{D}_H$
        if New Samples **then**:
            while $loss > loss_{TH}$ or $n < n_max$ **do**
                Sample N samples o, a from $\mathcal{D}_H$
                Sample $\hat{a} \sim \pi$
                Compute $loss = \frac{1}{N}\sum_i^N (\hat{a}_i - a_i)^2$
                Perform gradient descent update on π

consists of either executing actions provided by the agent or actions provided by the human. The agent reads an observation from the environment and an action is sampled based on the current policy. At any moment the human supervisor is able to override the agent's action by holding a joystick trigger. When this trigger is held, the actions performed by the human a_h are sent to the vehicle to be executed and are added to the human dataset $\mathcal{D}_H$ to update the policy according to the *Update Policy* subroutine.

The agent's policy π is a fully-connected, three hidden-layer, neural network with 130, 72, and 40 neurons, respectively. The network is randomly initialized with weights sampled from a normal distribution. The policy is optimized by minimizing the mean squared error loss using the

Root Mean Square Propagation (RMSProp) optimizer with learning rate of 1e-4. Unless defined otherwise, the human dataset $\mathcal{D}_H$ is initialized as an empty comma-separated value (CSV) file. Its main goal is to store the observations and actions performed by the human. The procedure to update the policy in real-time spawns a separate CPU thread to perform policy updates in real-time while the human either demonstrates the task or intervenes. This separate thread continuously checks for new demonstration or intervention data based on the size of the human dataset. If new samples are found, this thread samples a minibatch of 64 samples of observations and actions from the human dataset and is used to perform policy updates based on the mean squared error loss until it reaches the loss threshold of 0.005 or maximum number of epochs (in this case, 2000 epochs). This iterative update routine continues until the task performance threshold α is achieved, which can vary from task to task depending on the desired performance. For this work, we set α to 1 and only stop training after a pre-specified number of episodes defined in our experimentation methodology to empirically evaluate our method over a controlled number of human interactions, here defined as either human demonstrations or human interventions.

9.4.3 Experimental Methodology

Using the AirSim landing task, we tested our proposed CoL framework against several baseline conditions where we compared against using only a single human interaction modality (i.e. only demonstrations or only interventions) using equal amounts of human interaction time for each condition. By controlling for the human interaction time, we can assess if our method of utilizing multiple forms of human interaction provides an improvement over a single form of interaction given the same amount of human effort.

Each human participant (n=4) followed the same experimental protocol: given an RGB video stream from the downward-facing camera, the participant controlled the continuous roll, pitch, yaw, and throttle of the vehicle using an Xbox One joystick to perform 4, 8, 12 and 20 complete episodes of the perching task for three experimental conditions: demonstrations only, interventions only, and demonstrations plus interventions with the CoL method, with each condition starting from a randomly initialized policy. For the CoL condition, participants performed an equal number

of demonstrations and interventions to match the total number of episodes for that condition. For example, given 4 episodes of training, our CoL approach would train with learning from demonstrations in the first 2 episodes and then switch to learning from interventions for the last 2 episodes. We compared this to learning from demonstration for all 4 episodes as well as learning from interventions for all 4 episodes. This was repeated for 8, 12 and 20 episodes to study the effect of varying amounts of human interaction on task performance. Following the diagram in Figure 9.2 and Update Policy procedure on Algorithm 6, the agent's policy is trained on a separate thread in real-time, and a model is saved for each complete episode together with the human-observed states and the actions they performed. These saved models are later evaluated to assess task performance according to our evaluation procedure described in the next section.

We also compared our approach to a random agent as well as an agent trained using a state-of-the-art reinforcement learning approach. The reinforcement learning agent used a publicly available implementation of Proximal Policy Optimization (PPO) [67] with a four degree-of-freedom action space (pitch, roll, throttle, yaw) and was trained for 1000 episodes, using only task completion as a binary sparse reward signal. To investigate the effect of action-space complexity on task performance, we also implemented the PPO where only two actions (pitch and roll) and three actions (pitch, roll and throttle) were available; for both cases, all other actions were held to a constant value. For the two actions condition (pitch and roll), the agent was given constant throttle and descended in altitude at a constant velocity. For both conditions yaw was set to 0. Training time of the reinforcement learning agent was limited to the simulated environment running in real-time.

9.5 Numerical Results

We evaluated our method in terms of task completion percentage, defined as the number of times the drone successfully landed on the landing pad over 100 evaluation runs, for each training method as well as for different amounts of human training data. Additionally, We compared the number of human data samples, i.e. observation-action pairs, used during training for each condition.

Figure 9.4 compares the performance of the models trained using only interventions (Int), the models trained using only demonstrations (Demo), and the models trained using the Cycle-of-

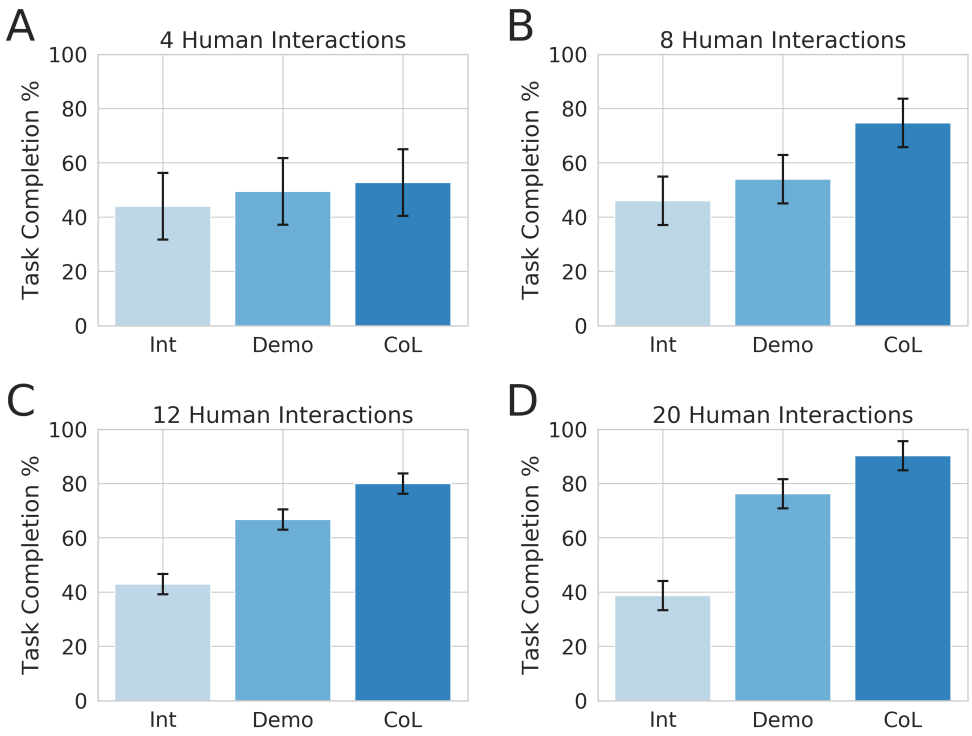


Figure 9.4: Performance comparison in terms of task completion with Interventions (Int), Demonstrations (Demo) and the Cycle-of-Learning (CoL) framework for (A) 4 human interactions, (B) 8 human interactions, (C) 12 human interactions and (D) 20 human interactions, respectively. Here, an interaction equates to a single demonstration or intervention and roughly corresponds to the number of episodes. Error bars denote 1 standard error of the mean. We see that CoL outperforms Int and Demo across nearly all human interaction levels. Reprinted from [8].

Learning approach (CoL). We show results for only these conditions as the random policy condition and the RL condition trained using PPO with the full four degree-of-freedom action space were not successful given the small amount of training episodes, as explained later in this section. Barplots show the task completion performance from each condition averaged over all participants with error bars representing 1 standard error. Subpanels show the performance for varying amounts of human interaction: 4, 8, 12 and 20 episodes. For the 4 human interaction condition (Figure 9.4A), all methods show similar task completion conditions. However, for the 8, 12 and 20 human interaction conditions, we see that the CoL approach achieves higher task completion percentages compared to the demonstration-only and intervention-only conditions, with the intervention condition performing the worst.

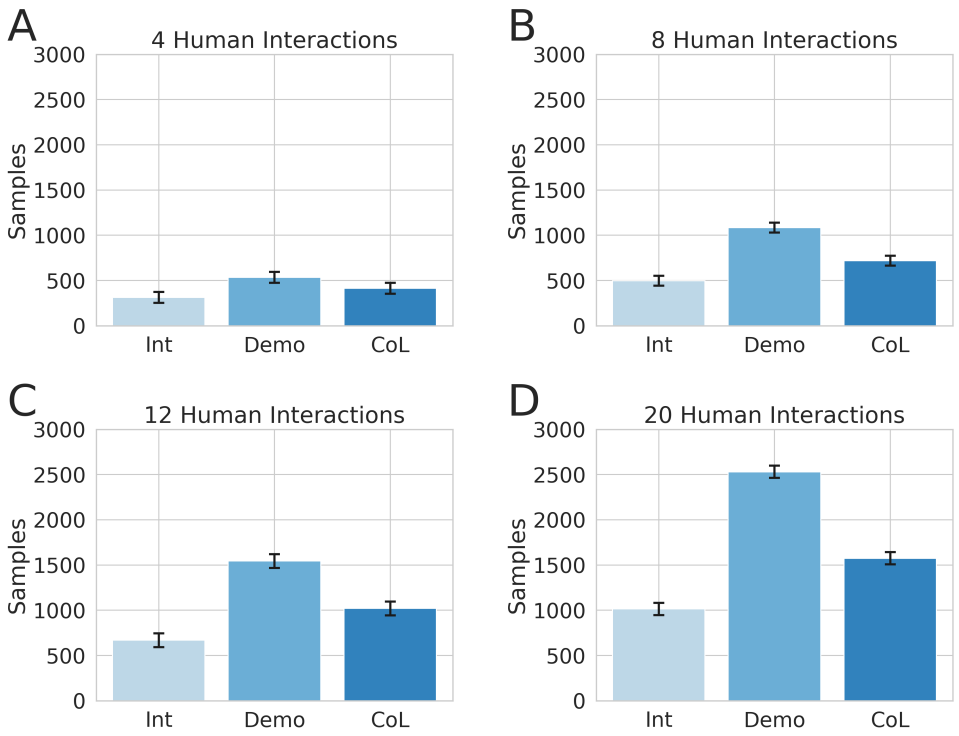


Figure 9.5: Comparison of the number of human samples used for training with Interventions (Int), Demonstrations (Demo) and the Cycle-of-Learning (CoL) framework for (A) 4 human interactions, (B) 8 human interactions, (C) 12 human interactions and (D) 20 human interactions, respectively. Error bars denote 1 standard error of the mean. We see that CoL uses less data than the demonstration-only condition and only slightly more data than the intervention-only condition. Reprinted from [8].

For the final condition of 20 episodes our proposed approach achieves 90.25% ($\pm$ 5.63% std. error) task completion as compared to 76.25% ($\pm$ 2.72% std. error) task completion using only demonstrations. In comparison, for the 8 episodes condition, our proposed approach achieves 74.75% ($\pm$ 9.38% std. error) task completion in contrast to 54.00% ($\pm$ 8.95% std. error) task completion when using only demonstrations.

Figure 9.5 compares the number of human data samples used to train the models for the same conditions and datasets as in Figure 9.4. For the final condition of 20 episodes our proposed approach used on average 1574.50 ($\pm$ 54.22 std. error) human-provided samples, which is 37.79% fewer data samples when compared to using only demonstrations. Note that the policies generated from this sparser dataset were able to increase task completion by 14.00%. These results yield a

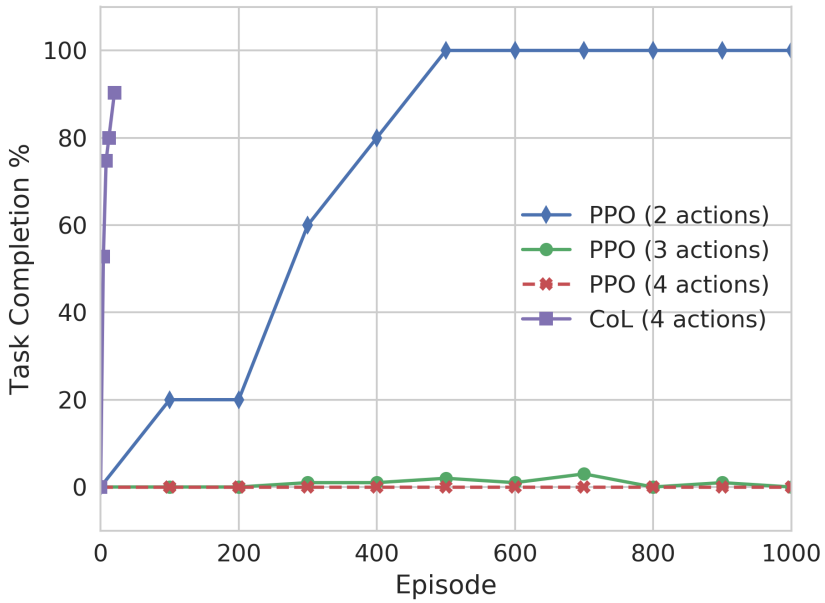


Figure 9.6: Performance comparison between the Cycle-of-Learning (CoL) with four continuous actions and the Deep Reinforcement Learning algorithm Proximal Policy Optimization (PPO) trained for three different task complexities using 2, 3, and 4 continuous actions. Reprinted from [8].

CoL agent that has 1.90 times the rate of task completion performance per sample when compared to learning from demonstrations alone. This value is computed by comparing the ratios of task completion rate to data samples utilized between the CoL agent and the demonstration-only agent, respectively. Averaging the results over all presented conditions and datasets, the task completion increased by 12.81% ($\pm$ 3.61% std. error) using 32.05% ($\pm$ 3.25% std. error) less human samples, which results in a CoL agent that overall has a task completion rate per sample 1.84 times higher than its counterparts.

Figure 9.6 shows a comparison between the performance of the CoL method as well as the PPO baseline comparisons using a 2, 3 and 4 degree-of-freedom action space. For PPO with two actions, the agent was able to achieve 100% task completion after 500 episodes, on average. However, when the action space complexity increases to three actions, the PPO agent performance was significantly reduced, now completing the task less than 5% of the time after training for 1000 episodes. As

expected, the PPO agent with the full four degree-of-freedom action space fails to complete the task after training for 1000 episodes. In contrast, the CoL method, with the same four degree-of-freedom action space, achieves about 90% task completion in only 20 episodes, representing significant gains in sample efficiency compared to PPO.

9.6 Summary

Learning from demonstrations in combination with learning from interventions yields a more proficient policy based on less data, when compared to either approach in isolation. It is likely that the superior performance of the CoL is due to combining the two methods in sequence so as to leverage their strengths while attenuating their deficiencies.

Having been initialized to a random policy, learning from interventions alone produced more random behaviors, making convergence to a baseline behavior much slower. Overall performance is thus slower to develop, resulting in lower percent completion for the same number of interaction episodes. Conversely, learning from demonstrations alone was quicker to converge to stable behavior, but it was consistently outperformed by the CoL across varying numbers of interactions, while having more training data to utilize. This seems initially counter-intuitive as more training data should result in a more accurate and presumably more proficient policy. However, as the demonstrations typically follow stable trajectories, the agent is less likely to encounter regions of the state space outside these trajectories. When enacting the policy at test time, any deviations from these previously observed states is not captured well in the policy, resulting in poor generalization performance. By allowing the agent to act under its current policy, in conjunction with adaptively updating the policy with corrective human-provided actions needed to recover from potentially catastrophic states, the dataset and subsequent policy is improved. Thus, the CoL allows for both rapid convergence to baseline stable behavior and then safe exploration of state space to make the policy more refined *and* robust.

The results shown in Figure 9.5 confirm the expectation that the combination of learning from demonstrations and interventions requires less data than the condition of learning from demonstrations alone, for the same number of episodes. This supports the notion that the CoL is

a more data efficient approach to training via human inputs. When additionally considering the superior performance exhibited in Figure 9.4, the data efficiency provided by this technique is even more significant. This result further supports the notion that a combinatorial learning strategy inherently samples more data rich inputs from the human observer.

It should be emphasized here that rather than providing an incremental improvement to a specific demonstration or intervention learning strategy, this work proposes an algorithmically agnostic methodology for combining modes of human-based learning. The primary assertion of this work is that learning is made more robust, data efficient, and safe through a fluid and complementary cycling of these two modes, and would similarly be improved with the addition of the later stages of the CoL (i.e. learning from human evaluation and reinforcement learning).

As seen in Figure 9.6, the PPO baseline comparison method was tested across varying complexity with different numbers of action dimensions. A striking result that can be seen is the significant drop-off in performance when going from two-actions, in which the drone had a constant downward throttle and only controlled roll and pitch, to three and four actions, in which the drone also had to control its own throttle. An obvious characteristic of a successful policy for the perching task is that the drone needs to descend in a stable and smooth manner, which is already provided in the two-action condition, as the downward throttle was set *apriori*. This makes the task of solving for an optimal policy much simpler. In the three and four action condition, however, this behavior must be learned from a sparse reward signal (success or failure to land), which is very difficult given limited episodes.

When implementing the CoL in real world environments, catastrophic failures may be seriously damaging to the autonomous agent, and thus unacceptable. Having a human observer capable of intervening provides a mechanism to prevent this inadmissible outcome. Further, techniques that might be applied to enforce a similar level of safety automatically might limit the exploration of the state space, yielding a less robust or less capable policy. Analogously to the shift of policy design from roboticists or domain experts to human users and laypersons, which is yielded by human-in-the-loop learning, the technique of learning from interventions shifts the implementation

of system fail safes away from developers toward users. This shift leverages human abilities to predict outcomes, adapt to dynamic circumstances, and synthesize contextual information in decision making.

9.6.1 Current Limitations and Future Directions

Our current implementation is limited to the first two stages of the CoL: learning from demonstrations and interventions. Our planned future work will include adding in more components of the CoL; for example, learning from human evaluative feedback as done in [46, 47, 48]. Additionally, we aim to incorporate reinforcement learning techniques to further fine-tune the learning performance after learning from human demonstrations, interventions and evaluations using an actor-critic style RL architecture [157].

A second limitation of the current implementation is that it requires the human to supervise the actions taken by the agent at all times. Future work aims to incorporate confidence metrics in our learning models so that the autonomous system can potentially halt its own actions when it determines it has low confidence and query the human directly for feedback in a mixed-initiative style framework [43], similar to active learning techniques. Furthermore, our results clearly indicate that a two-stage process - with a primary stage with a large proportion of human-provided actions followed by a secondary stage with a smaller proportion of those actions - outperforms processes with uniformly large or small amounts of human data throughout. This suggests there is perhaps an optimal point in the learning process at which to vary in the amount of human input from full demonstrations to interventions. Figure 9.4 illustrates this notion across the varying number of interactions shown in the subfigures, i.e. through the change in relative performance between the three conditions. In future work we will examine if such an optimal mixture or sequencing of demonstrations and interventions exists, such that learning speed and stability are maximized, and if so, whether it is operator dependent. Rather than having a predetermined staging of the demonstrations and interventions that is potentially suboptimal, a mixed initiative framework could determine this optimal transition point. This could further reduce the burden on the human observer, allow for faster training, and even provide a mechanism to generate more robust policies through

guided exploration of the state space.

This work demonstrates the first two stages of the CoL in a simulation environment with the goal of eventually transitioning to physical systems, such as an sUAS. The CoL framework was implicitly designed for use in real world systems, where interactions are limited, and catastrophic actions are unacceptable. As can be seen in Figure 9.6, our method learns to perform the perching task in several orders of magnitude less time than traditional RL approaches, potentially allowing for feasible on-the-fly training of real systems. Therefore, we expect that the application of the CoL to sUAS platforms, or other physical systems, should operate in effectively the same manner as demonstrated in this work. Future efforts will focus on transitioning this framework onto such physical platforms to study its efficacy in real world settings. One critical hurdle that must be overcome, is the implementation of the learning architecture on embedded hardware, constrained by the limited payload of an sUAS.

Additionally, given that we are utilizing a relatively high fidelity simulation environment, i.e. AirSim, it may be beneficial to bootstrap a real world system with a policy learned in simulation. Although there are numerous challenges in transferring a policy learned in simulation into the real world, the CoL itself should allow for significantly smoother transfer due to its cyclic nature in which the user can revert to more direct and user intensive inputs at any point during the learning to allow for adaptation to previously unobserved states. This capability inherently provides a method of transfer learning in the case of disparities between simulated and real world properties of the vehicle, sensors, and environment. For example, if the perching behavior learned in simulation was transferred to an actual sUAS, the vehicle dynamics may have unmodeled non-linearities, the imagery may have dynamic range limitations, or the environment may present exogenous gust disturbances. In such cases, the baseline policy would be monitored and corrected via learning from intervention, if these discrepancies yielded undesirable or possibly catastrophic behaviors.

10. INTEGRATING BEHAVIOR CLONING AND REINFORCEMENT LEARNING FOR IMPROVED PERFORMANCE IN SPARSE AND DENSE REWARD ENVIRONMENTS*

This paper investigates how to efficiently transition and update policies, trained initially with demonstrations, using off-policy actor-critic reinforcement learning. It is well-known that techniques based on Learning from Demonstrations, for example behavior cloning, can lead to proficient policies given limited data. However, it is currently unclear how to efficiently update that policy using reinforcement learning as these approaches are inherently optimizing different objective functions. Previous works have used loss functions, which combine behavior cloning losses with reinforcement learning losses to enable this update. However, the components of these loss functions are often set anecdotally, and their individual contributions are not well understood. In this work, we propose the Cycle-of-Learning (CoL) framework that uses an actor-critic architecture with a loss function that combines behavior cloning and 1-step Q-learning losses with an off-policy pre-training step from human demonstrations. This enables transition from behavior cloning to reinforcement learning without performance degradation and improves reinforcement learning in terms of overall performance and training time. Additionally, we carefully study the composition of these combined losses and their impact on overall policy learning. We show that our approach outperforms state-of-the-art techniques for combining behavior cloning and reinforcement learning for both dense and sparse reward scenarios. Our results also suggest that directly including the behavior cloning loss on demonstration data helps to ensure stable learning and ground future policy updates.

*Adapted permission from "Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Sparse and Dense Reward Environments", by Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich, presented at the 2020 International Conference on Autonomous Agents and Multi-Agent Systems [9] and is partially reproduced here, Copyright 2020 by the International Foundation for Autonomous Agents and MultiAgent Systems.

10.1 Problem Definition

Reinforcement Learning (RL) has yielded many recent successes in solving complex tasks that meet and exceed the capabilities of human counterparts, demonstrated in video game environments [76], robotic manipulators [224], and various open-source simulated scenarios [225]. However, these RL approaches are sample inefficient and slow to converge to this impressive behavior, limited significantly by the need to explore potential strategies through trial and error, which produces initial performance significantly worse than human counterparts. The resultant behavior that is initially random and slow to reach proficiency is poorly suited to various situations, such as physically embodied ground and air vehicles or in scenarios where sufficient capability must be achieved in short time spans. In such situations, the random exploration of the state space of an untrained agent can result in unsafe behaviors and catastrophic failure of a physical system, potentially resulting in unacceptable damage or downtime. Similarly, slow convergence of the agent's performance requires exceedingly many interactions with the environment, which is often prohibitively difficult or infeasible for physical systems that are subject to energy constraints, component failures, and operation in dynamic or adverse environments. These sample efficiency pitfalls of RL are exacerbated even further when trying to learn in the presence of sparse rewards, often leading to cases where RL can fail to learn entirely.

One approach for overcoming these limitations is to utilize demonstrations of desired behavior from a human data source (or potentially some other agent) to initialize the learning agent to a significantly higher level of performance than is yielded by a randomly initialized agent. This is often termed Learning from Demonstrations (LfD) [226], which is a subset of imitation learning that seeks to train a policy to imitate the desired behavior of another policy or agent. LfD leverages data (in the form of state-action tuples) collected from a demonstrator for supervised learning, and can be used to produce an agent with qualitatively similar behavior in a relatively short training time and with limited data. This type of LfD, called Behavior Cloning (BC), learns a mapping between the state-action pairs contained in the set of demonstrations to mimic the behavior of the demonstrator.

Though BC techniques do allow for the relatively rapid learning of behaviors that are comparable to that of the demonstrator, they are limited by the quality and quantity of the demonstrations provided and are only improved by providing additional, high-quality demonstrations. In addition, BC is plagued by the distributional drift problem in which a mismatch between the learned policy distribution of states and the distribution of states in the training set can cause errors that propagate over time and lead to catastrophic failures. By combining BC with subsequent RL, it is possible to address the drawbacks of either approach, initializing a significantly more capable and safer agent than with random initialization, while also allowing for further self-improvement without needing to collect additional data from a human demonstrator. However, it is currently unclear how to effectively update a policy initially trained with BC using RL as these approaches are inherently optimizing different objective functions. Previous works have used loss functions that combine BC losses with RL losses to enable this update, however, the components of these loss functions are often set anecdotally and their individual contributions are not well understood.

In this work, we propose the Cycle-of-Learning (CoL) framework, which uses an actor-critic architecture with a loss function that combines behavior cloning and 1-step Q-learning losses with an off-policy algorithm, and a pre-training step to learn from human demonstrations. Unlike previous approaches to combine BC with RL, such as [227], our approach uses an actor-critic architecture to learn both a policy and value function from the human demonstration data, which we show, speeds up learning. Additionally, we perform a detailed component analysis of our method to investigate the individual contributions of pre-training, combined losses, and sampling methods of the demonstration data and their effects on transferring from BC to RL. To summarize, the main contribution of this work are:

- We introduce an actor-critic based method, that combines pre-training as well as combined loss functions to learn both a policy and value function from demonstrations, to enable transition from behavior cloning to reinforcement learning.
- We show that our method can transfer from BC to RL without performance degradation while improving upon existing state-of-the-art BC to RL algorithms in terms of overall performance

and training time.

- We perform a detailed analysis to investigate the contributions of the individual components in our method.

Our results show that our approach outperforms BC, Deep Deterministic Policy Gradients (DDPG), and Demonstration Augmented Policy Gradient (DAPG) in two different application domains for both dense- and sparse-reward settings. Our results also suggest that directly including the behavior cloning loss on demonstration data helps to ensure stable learning and ground future policy updates, and that a pre-training step enables the policy to start at a performance level greater than behavior cloning.

10.2 Preliminaries

We adopt the standard Markov Decision Process (MDP) formulation for sequential decision making [157], which is defined as a tuple (S, A, R, P, γ), where S is the set of states, A is the set of actions, $R(s, a)$ is the reward function, $P(s'|s, a)$ is the transition probability function and γ is a discount factor. At each state $s \in S$, the agent takes an action $a \in A$, receives a reward $R(s, a)$ and arrives at state s' as determined by $P(s'|s, a)$. The goal is to learn a behavior policy π which maximizes the expected discounted total reward. This is formalized by the Q-function, sometimes referred to as the state-action value function:

$$Q^{\pi}(s, a) = \mathbb{E}_{a_t \sim \pi}\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)\right]$$

taking the expectation over trajectories obtained by executing the policy π starting at $s_0 = s$ and $a_0 = a$.

Here we focus on actor-critic methods which seek to maximize

$$J(\theta) = \mathbb{E}_{s \sim \mu}[Q^{\pi(.|\theta)}(s, \pi(s|\theta))]$$

with respect to parameters θ and an initial state distribution μ. The Deep Deterministic Policy

Gradient (DDPG) [225] is an off-policy actor-critic reinforcement learning algorithm for continuous action spaces, which calculates the gradient of the Q-function with respect to the action to train the policy. DDPG makes use of a replay buffer to store past state-action transitions and target networks to stabilize Q-learning [76]. Since DDPG is an off-policy algorithm, it allows for the use of arbitrary data, such as demonstrations from another source, to update the policy. A demonstration trajectory is a tuple (s, a, r, s') of state s, action a, the reward $r = R(s, a)$ and the transition state s' collected from a demonstrator's policy. In most cases these demonstrations are from a human observer, although in principle these demonstrations can come from any existing agent or policy.

10.3 Related Work

Several works have shown the efficacy of combining behavior cloning with reinforcement learning across a variety of tasks. Recent work by [117], known as Deep Q-learning from Demonstrations (DQfD), combined behavior cloning with deep Q-learning [76] to learn policies for Atari games by leveraging a loss function that combines a large-margin supervised learning loss function, 1-step Q-learning loss, and an n-step Q-learning loss function that helps ensure the network satisfies the Bellman equation. This work was extended to continuous action spaces by [118] with DDPG from Demonstrations (DDPGfD), who proposed an extension of DDPG [225] that uses human demonstrations, and applied their approach to object manipulation tasks for both simulated and real robotic environments. The loss functions for these methods include the n-step Q-learning loss, which is known to require on-policy data to accurately estimate. Similar work by [228] combined behavior cloning-based demonstration learning, goal-based reinforcement learning, and DDPG for robotic manipulation of objects in a simulated environment.

A method that is very similar to ours is the Demonstration Augmented Policy Gradient (DAPG) [227], a policy-gradient method that uses behavior cloning as a pre-training step together with an augmented loss function with a heuristic weight function that interpolates between the policy gradient loss, computed using the Natural Policy Gradient [229], and behavior cloning loss. They apply their approach across four different robotic manipulations tasks using a 24 Degree-of-Freedom (DoF) robotic hand in a simulator and show that DAPG outperforms DDPGfD [118] across all tasks.

Their work also showed that behavior cloning combined with Natural Policy Gradient performed very similarly to DAPG for three of the four tasks considered, showcasing the importance of using a behavior cloning loss both in pre-training and policy training.

10.4 Integrating Behavior Cloning and Reinforcement Learning

The Cycle-of-Learning (CoL) framework is a method for leveraging multiple modalities of human input to improve the training of RL agents. These modalities can include human demonstrations, i.e. human-provided exemplar behaviors, human interventions, i.e. interdictions in agent behavior with subsequent partial demonstrations, and human evaluations, i.e. sparse indications of the quality of agent behavior. These individual mechanisms of human interaction have been previously shown to provide various benefits in learning performance and efficiency [230, 231, 232, 8, 233]. The successful integration of these disparate techniques, which would leverage their complementary characteristics, requires a learning architecture that allows for optimization of common objective functions and consistent representations. An actor-critic framework with a combined loss function, as presented in this work, is such an architecture.

In this paper, we focus on extending the Cycle-of-Learning framework to tackle the known issue of transitioning BC policies to RL by utilizing an actor-critic architecture with a combined BC+RL loss function and pre-training phase for continuous state-action spaces, that can learn in both dense- and sparse-reward environments. The main advantage of our method is the use of an off-policy, actor-critic architecture to pre-train both a policy and value function, as well as continued re-use of demonstration data during agent training, which reduces the amount of interactions needed between the agent and environment. This is an important aspect especially for robotic applications or real-world systems where interactions can be costly.

The combined loss function consists of the following components: an expert behavior cloning loss that drives the actor's actions toward previous human trajectories, 1-step return Q-learning loss to propagate values of human trajectories to previous states, the actor loss, and a L_2 regularization loss on the actor and critic to stabilize performance and prevent over-fitting during training. The implementation of each loss component and their combination are defined as follows:

- **Expert behavior cloning loss ($\mathcal{L}_{BC}$):** Given expert demonstration subset $\mathcal{D}_E$ of continuous states and actions s^E and a^E visited by the expert during a task demonstration over T time steps

$$\mathcal{D}_E = \left\{ s_0^E, a_0^E, s_1^E, a_1^E, ..., s_T^E, a_T^E \right\}, \tag{10.1}$$

a behavior cloning loss (mean squared error) from demonstration data $\mathcal{L}_{BC}$ can be written as

$$\mathcal{L}_{BC}(\theta_\pi) = \frac{1}{2} \left(\pi(s_t|\theta_\pi) - a_t^E) \right)^2 \tag{10.2}$$

in order to minimize the difference between the actions predicted by the actor network $\pi(s_t)$, parametrized by θ_π, and the expert actions a_{E_t} for a given state vector s_t.

- **1-step return Q-learning loss ($\mathcal{L}_1$):** The 1-step return R_1 can be written in terms of the critic network Q, parametrized by θ_Q, as

$$R_1 = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta_\pi)|\theta_Q). \tag{10.3}$$

In order to satisfy the Bellman equation, we minimize the difference between the predicted Q-value and the observed return from the 1-step roll-out for a batch of sampled states **s**:

$$\mathcal{L}_{Q_1}(\theta_Q) = \frac{1}{2} \left(R_1 - Q(\mathbf{s}, \pi(\mathbf{s}|\theta_\pi)|\theta_Q) \right)^2. \tag{10.4}$$

- **Actor Q-loss ($\mathcal{L}_A$):** It is assumed that the critic function Q is differentiable with respect to the action. Since we want to maximize the Q-values for the current state, the actor loss became the negative of the Q-values predicted by the critic for a batch of sampled states **s**:

$$\mathcal{L}_A(\theta_\pi) = -Q(\mathbf{s}, \pi(\mathbf{s}|\theta_\pi)|\theta_Q). \tag{10.5}$$

Combining the above loss functions for the Cycle-of-Learning becomes

$$\mathcal{L}_{CoL}(\theta_Q, \theta_\pi) = \lambda_{BC}\mathcal{L}_{BC}(\theta_\pi) + \lambda_A\mathcal{L}_A(\theta_\pi) + \lambda_{Q_1}\mathcal{L}_{Q_1}(\theta_Q) + \lambda_{L2Q}\mathcal{L}_{L2}(\theta_Q) + \lambda_{L2\pi}\mathcal{L}_{L2}(\theta_\pi). \tag{10.6}$$

Our approach starts by collecting contiguous trajectories from expert policies and stores the current and subsequent state-actions pairs, reward received, and task completion signal in a permanent expert memory buffer $\mathcal{D}_E$. During the pre-training phase, the agent samples a batch of trajectories from the expert memory buffer $\mathcal{D}_E$ containing expert trajectories to perform updates on the actor and critic networks using the same combined loss function (Equations 10.6). This procedure shapes the actor and critic initial distributions to be closer to the expert trajectories and eases the transition from policies learned through expert demonstration to reinforcement learning.

After the pre-training phase, the policy is allowed to roll-out and collect its first on-policy samples, which are stored in a separate first-in-first-out memory buffer with only the agent's samples. After collecting a given number of on-policy samples, the agent samples a batch of trajectories comprising 25% of samples from the expert memory buffer and 75% from the agent's memory buffer. This fixed ratio guarantees that each gradient update is grounded by expert trajectories. If a human demonstrator is used, they can intervene at any time the agent is executing their policy, and add this new trajectories to the expert memory buffer.

The proposed method is shown in Algorithm 6.

10.5 Numerical Results

10.5.1 Experimental Setup

As described in the previous sections, in our approach, the Cycle-of-Learning (CoL), we collect contiguous trajectories from expert policies and store them in a permanent memory buffer. The policy is allowed to roll-out and is trained with a combined loss from a mix of demonstration and agent data, stored in a separate first-in-first-out buffer. We validate our approach in three

Algorithm 6 Cycle-of-Learning (CoL): Transitioning from Demonstration to Reinforcement Learning

Input:

Environment env, number of training steps T, number of training steps per batch M, number of pre-training steps L, number of gradient updates K, and CoL hyperparameters λ_{Q_1}, λ_{BC}, λ_A, λ_{L2Q}, $\lambda_{L2\pi}$, τ.

Output:

Trained actor $\pi(s|\theta_\pi)$ and critic $Q(s, \pi|\theta_Q)$ networks.

Randomly initialize:

Actor network $\pi(s|\theta_\pi)$ and its target $\pi'(s|\theta_{\pi'})$.

Critic network $Q(s, \pi|\theta_Q)$ and its target $Q'(s, \pi'|\theta_{Q'})$.

Initialize agent and expert replay buffers $\mathcal{R}$ and $\mathcal{R}_E$.
Load $\mathcal{R}$ and $\mathcal{R}_E$ with expert dataset $\mathcal{D}_E$.
for pre-training steps = 1, ..., L **do**
    Call *TrainUpdate*() procedure.
for training steps = 1, ..., T **do**
    Reset env and receive initial state s_0.
    for batch steps = 1, ..., M **do**
        Select action $a_t = \pi(s_t|\theta_\pi)$ according to policy.
        Perform action a_t and observe reward r_t and next state s_{t+1}.
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$.
    for update steps = 1, ..., K **do**
        Call *TrainUpdate*() procedure.
procedure TRAINUPDATE()
    if Pre-training **then**
        Randomly sample N transitions (s_i, a_i, r_i, s_{i+1}) from the expert replay buffer $\mathcal{R}_E$.
    else
        Randomly sample $N * 0.25$ transitions (s_i, a_i, r_i, s_{i+1}) from the expert replay buffer $\mathcal{R}_E$ and $N * 0.75$ transitions from the agent replay buffer $\mathcal{R}$.
    Compute $\mathcal{L}_{Q_1}(\theta_Q)$, $\mathcal{L}_{BC}(\theta_\pi)$, $\mathcal{L}_A(\theta_\pi)$, $\mathcal{L}_{L2}(\theta_Q)$, $\mathcal{L}_{L2}(\theta_\pi)$
    Update actor and critic for K steps according to Equation 10.6.
    Update target networks:

$$\theta_{\pi'} \leftarrow \tau\theta_\pi + (1-\tau)\theta_{\pi'},$$
$$\theta_{Q'} \leftarrow \tau\theta_Q + (1-\tau)\theta_{Q'}.$$

environments with continuous observation- and action-space: LunarLanderContinuous-v2 [209] (dense and sparse reward cases) and a custom quadrotor landing task [8] implemented using

Microsoft AirSim [210]. The dense reward case of LunarLanderContinuous-v2 is the standard environment provided by OpenAI Gym library [209]: the state space consists of a eight-dimensional continuous vector with inertial states of the lander, the action space consists of a two-dimensional continuous vector controlling main and side thrusters, and the reward is given at every step based on the relative motion of the lander with respect to the landing pad (bonus reward is given when the landing is completed successfully). The sparse reward case is a custom modification with the same reward scheme and state-action space, however the reward is stored during the policy roll-out and is only given to the agent when the episode ends and is zero otherwise. The custom quadrotor landing task is a modified version of the environment proposed by Goecks et al. [8], implemented using Microsoft AirSim [210], which consists of landing a quadrotor on a static landing pad in a simulated gusty environment, as seen in Figure 9.3. The state space consists of a fifteen-dimensional continuous vector with inertial states of the quadrotor and visual features that represent the landing pad image-frame position and radius as seen by a downward-facing camera. The action space is a four-dimensional continuous vector that sends velocity commands for throttle, roll, pitch, and yaw. Wind is modeled as noise applied directly to the actions commanded by the agent and follows a temporal-based, instead of distance-based, discrete wind gust model [234] with 65% probability of encountering a wind gust at each time step. This was done to induce additional stochasticity in the environment. The gust duration is uniformly sampled to last between one to three real time seconds and can be imparted in any direction, with maximum velocity of half of what can be commanded by the agent along each axis. This task has a sparse-reward scheme (reward R is given at the end of the episode, and is zero otherwise) based on the relative distance r_{rel} between the quadrotor and the center of the landing pad at the final time step of the episode:

$$R = \frac{1}{1 + r_{rel}^2}.$$

The hyperparameters used in CoL for each environment are described in table 10.1.

The baselines that we compare our approach to are Deep Deterministic Policy Gradient (DDPG)

Table 10.1: Cycle-of-Learning hyperparemeters for each environment: (a) LunarLanderContinuous-v2 and (b) Microsoft AirSim. Reprinted from [9].

	Environments	
Hyperparameter	(a)	(b)
λ_{Q_1} factor	1.0	1.0
λ_{BC} factor	1.0	1.0
λ_A factor	1.0	1.0
λ_{L2Q} factor	$1.0e^{-5}$	$1.0e^{-5}$
$\lambda_{L2\pi}$ factor	$1.0e^{-5}$	$1.0e^{-5}$
Batch size	512	512
Actor learning rate	$1.0e^{-3}$	$1.0e^{-3}$
Critic learning rate	$1.0e^{-4}$	$1.0e^{-4}$
Memory size	$5.0e^{5}$	$5.0e^{5}$
Expert trajectories	20	5
Pre-training steps	$2.0e^{4}$	$2.0e^{4}$
Training steps	$5.0e^{6}$	$5.0e^{5}$
Discount factor γ	0.99	0.99
Hidden layers	3	3
Neurons per layer	128	128
Activation function	ELU	ELU

[225, 235], Demonstration Augmented Policy Gradient (DAPG) [227], and traditional behavior cloning (BC). For the DDPG baseline we used an open-source implementation by Stable Baselines [236]. The hyperparameters used concur with the original DDPG publication [225]: actor and critic networks with 2 hidden layers with 400 and 300 units respectively, optimized using Adam [237] with learning rate of 10^{-4} for the actor and 10^{-3} for the critic, discount factor of $\gamma = 0.99$, trained with minibatch size of 64, and replay buffer size of 10^6. Exploration noise was added to the action following an Ornstein-Uhlenbeck process [238] with mean of 0.15 and standard deviation of 0.2. For the DAPG baseline we used an official release of the DAPG codebase from the authors [†]. The policy is represented by a deep neural network with three hidden layers of 128 units each, pre-trained with behavior cloning for 100 epochs, with a batch size of 32 samples, and learning rate of 10^{-3}, $\lambda_0 = 0.01$, and $\lambda_1 = 0.99$. The BC policies are trained by minimizing the mean squared

[†]Code available at `https://github.com/aravindr93/hand_dapg` [239].

error between the expert demonstrations and the output of the model. The policies consist of a fully-connected neural network with 3 hidden layers with 128 units each and exponential linear unit (ELU) activation function [240]. The BC policy was evaluated for 100 episodes which was used to calculate the mean and standard error of the performance of the policy.

All baselines that rely on demonstrations, namely BC, DAPG, and CoL, use the same human trajectories collected in the LunarLanderContinuous-v2 and custom Microsoft AirSim environment.

10.5.2 Experimental Results

The comparative performances of the CoL against the baseline methods (BC, DDPG and DAPG) for the LunarLanderContinuous-v2 environment are presented via their training curves in Figure 10.1, using the standard dense reward. The mean reward of the BC pre-trained from the human demonstrations is also shown for reference, and its standard error is shown by the shaded band. The CoL reward initializes to values at or above the BC and steadily improves throughout the reinforcement learning phase. Conversely, the DDPG RL baseline initially returns rewards lower than the BC and slowly improves until its performance reaches similar levels to the CoL after approximately one million steps. However, this baseline never performs as consistently as the CoL and eventually begins to diverge, losing much of its performance gains after about four million steps. The DAPG baseline initial performance, similar to the CoL, surpasses behavior cloning due to the pre-training phase and slowly converges to a high score, although slower than the CoL.

When using sparse rewards, meaning the rewards generated by the LunarLanderContinuous-v2 environment are provided only at the last time step of each episode, the performance improvement of the CoL relative to the DDPG and DAPG baselines is even greater (Figure 10.2). The performance of the CoL is qualitatively similar during training to that of the dense case, with an initial reward roughly equal to or greater than that of the BC and a consistently increasing reward. Conversely, the performance of the DDPG baseline is greatly diminished for the sparse reward case, yielding effectively no improvement throughout the whole training period. The training of the DAPG does not deteriorate when compared to the dense reward case but still the performance does not match CoL for the specified training time.

The results for the more realistic and challenging AirSim quadrotor landing environment (Figure 10.3) illustrate a similar trend. The CoL initially returns rewards above the BC, DDPG, and DAPG baselines and steadily increases its performance, with DAPG converging at end to a similar level of performance. The DDPG baseline practically never succeeds and subsequently fails to learn a viable policy, while displaying greater variance in performance when compared to CoL and DAPG. Noting that successfully landing on the target would generate a sparse episode reward of approximately 0.64, it is clear that these baseline algorithms, with exception of DAPG, rarely generate a satisfactory trajectory for the duration of training.

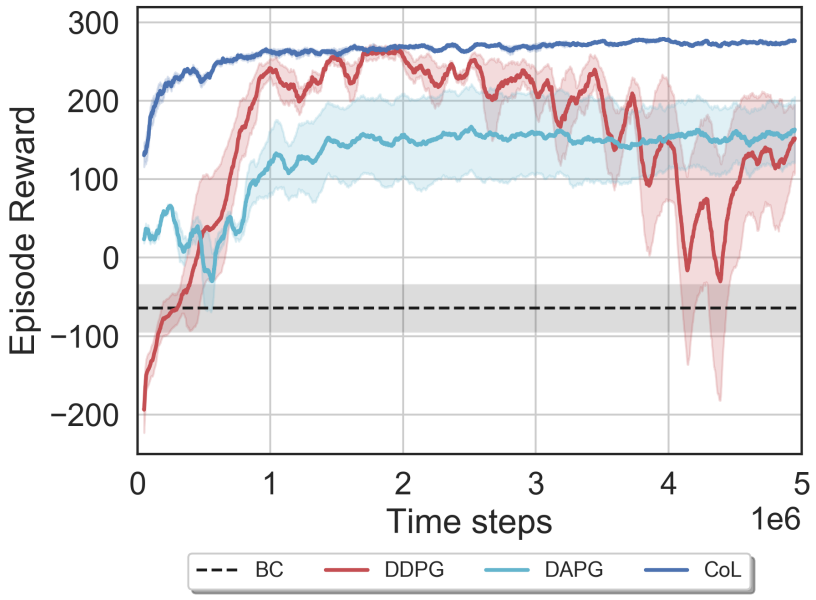


Figure 10.1: Comparison of CoL, BC, DDPG, and DAPG for 3 random seeds (bold line representing the mean and shaded area the standard error) in the dense–reward LunarLanderContinuous-v2 environment. Reprinted from [9].

10.5.3 Component Analysis

Several component analyses were performed to evaluate the impact of each of the critical elements of the CoL on learning. These respectively include the effects of pre-training, the

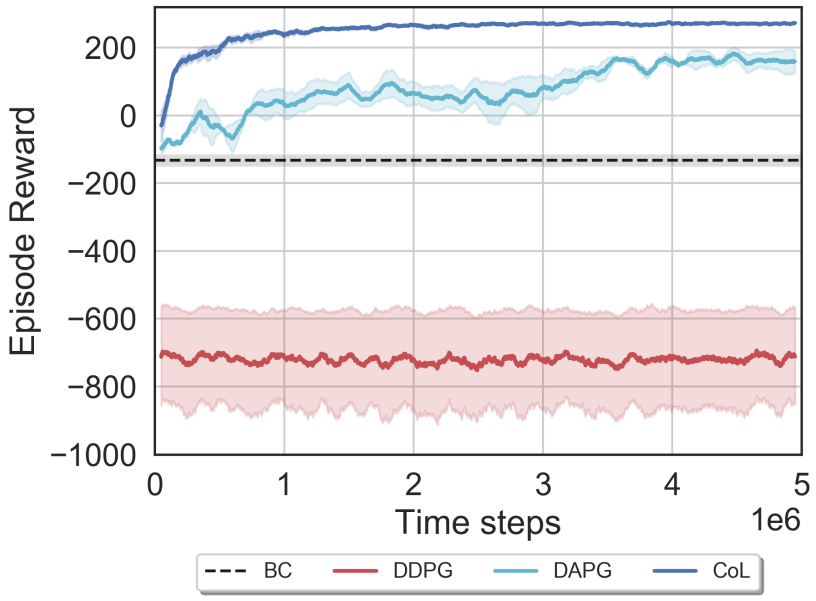


Figure 10.2: Comparison of CoL, BC, DDPG, and DAPG for 3 random seeds (bold line representing the mean and shaded area the standard error) in the sparse-reward LunarLanderContinuous-v2 environment. Reprinted from [9].

Table 10.2: Method Comparison on LunarLanderContinuous-v2 environment, dense-reward case. Reprinted from [9].

Method	**Pre-Training Loss**	**Training Loss**	**Buffer Type**	**Average Reward**
CoL	$\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$	$\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$	Fixed Ratio	261.80 $\pm$ 22.53
CoL-PT	None	$\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$	Fixed Ratio	253.24 $\pm$ 46.50
CoL+PER	$\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$	$\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$	PER	245.24 $\pm$ 37.66
DAPG	$\mathcal{L}_{BC}$	Augmented Policy Gradient	None	127.99 $\pm$ 37.28
DDPG	None	$\mathcal{L}_{Q_1} + \mathcal{L}_A$	Uniform	152.98 $\pm$ 69.45
BC	$\mathcal{L}_{BC}$	None	None	-48.83 $\pm$ 27.68*
BC+DDPG	$\mathcal{L}_{BC}$	$\mathcal{L}_{Q_1} + \mathcal{L}_A$	Uniform	-57.38 $\pm$ 50.11
CoL-BC	$\mathcal{L}_{Q1} + \mathcal{L}_A$	$\mathcal{L}_{Q_1} + \mathcal{L}_A$	Fixed Ratio	-105.65 $\pm$ 196.85

Summary of learning methods. Enumerated for each method are all non-zero loss components (excluding regularization), buffer type, and average and standard error of the reward throughout training (after pre-training) across the three seeds, evaluated with dense reward in LunarLander-Continuous-v2 environment. *For BC, these values are computed from 100 evaluation trajectories of the final pre-trained agent.

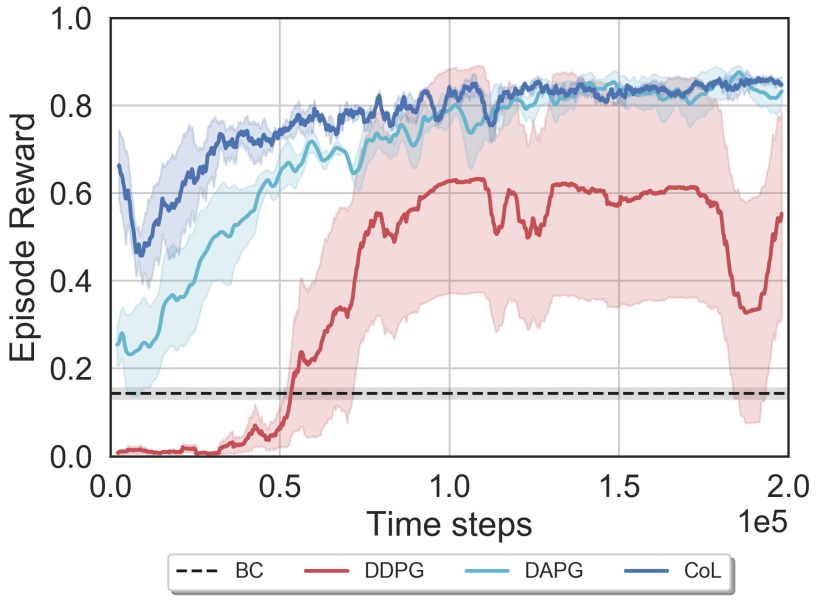


Figure 10.3: Comparison of CoL, BC, DDPG, and DAPG for 3 random seeds (bold line representing the mean and shaded area the standard error) in the sparse-reward Microsoft AirSim quadrotor landing environment. Reprinted from [9].

combined loss function, and the sample composition of the experience replay buffer. The results of each analysis are shown in Figures 10.4-10.6 and are summarized in Table 10.2.

10.5.3.1 Effects of Pre-Training

To determine the effects of pre-training on performance we compare the standard CoL against an implementation without this pre-training phase, where the number of pre-training steps $L = 0$, denoted as *CoL-PT*. The complete combined loss, as seen in Equations 10.6 is used during the reinforcement learning phase. This condition assesses the impact on learning performance of not pre-training the agent, while still using the combined loss in the RL phase. As seen in Figure 10.4, this condition differs from the baseline CoL in its initial performance being worse, i.e. significantly below the BC, but does reach similar rewards after several hundred thousand steps, exhibiting the same consistent response during training thereafter. Effectively, this highlights that the benefit of pre-training is improved initial response and significant speed gain in reaching steady-state performance level, without qualitatively impacting the long-term training behavior.

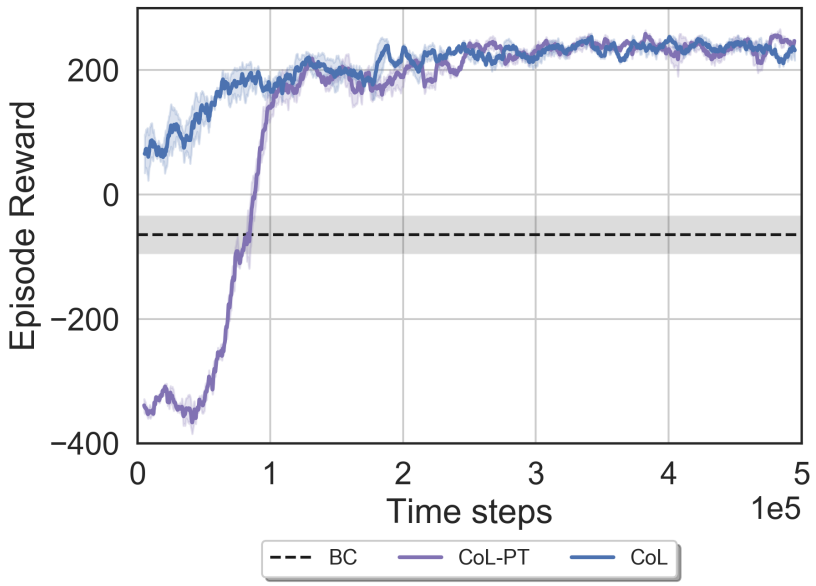


Figure 10.4: Effects of the pre-training phase in the Cycle-of-Learning. Results for 3 random seeds (bold line representing the mean and shaded area the standard error) showing component analysis in LunarLanderContinuous-v2 environment comparing pre-trained Cycle-of-Learning (CoL curve) against the Cycle-of-Learning without the pre-training phase (CoL-PT curve) and the behavior cloning (BC) baseline. Reprinted from [9].

10.5.3.2 Effects of Combined Loss

To determine the effects of the combined loss function on performance we compare the standard CoL against two alternate learning implementations: 1) the CoL without the behavioral cloning expert loss on the actor ($\lambda_{BC} := 0$) during both pre-training and RL phases, denoted as *CoL-BC*, and 2) standard BC followed by DDPG using standard loss functions, denoted as *BC+DDPG*. For the implementation of the CoL without the behavior cloning loss (*CoL-BC*), the critic loss remains the same as in Equation 10.6 for both training phases. This condition assesses the impact on learning performance of the behavior cloning loss component $\mathcal{L}_{BC}$, given otherwise consistent loss functions in both pre-training and RL phases. As seen in Figure 10.5, this condition (purple, dashed) improves upon the CoL-PT condition (Figure 10.4) in its initial reward return and similarly achieves comparable performance to the baseline CoL in the first few hundred thousand steps, but then steadily deteriorates as training continues, with several catastrophic losses in performance. This result makes clear that the behavioral cloning loss is an essential component of the combined loss function toward maintaining performance throughout training, anchoring the learning to some previously demonstrated behaviors that are sufficiently proficient.

The second of these comparative implementations that illustrate the effects of the combined loss, behavior cloning with subsequent DDPG (*BC+DDPG*), utilized standard loss functions (Equations 10.2, 10.4, and 10.5) rather than the CoL combined loss in both phases (Equation 10.6). Pre-training of the actor with BC uses only the regression loss, as seen in Equation 10.2. DDPG utilizes standard loss functions for the actor and critic, as seen in Equation 10.7. This condition assesses the impact on learning performance of standardized loss functions rather than our combined loss functions across both training phases. This condition (Figure 10.5; red, dashed) produces initial rewards below the BC response and subsequently improves in performance only to an average level similar to that of the BC and is much less stable in its response throughout training, as indicated by the wide standard error band. This result indicates that simply sequencing standard BC and RL algorithms results in significantly worse performance and stability even after millions of training steps, emphasizing the value of a consistent combined loss function across all training phases.

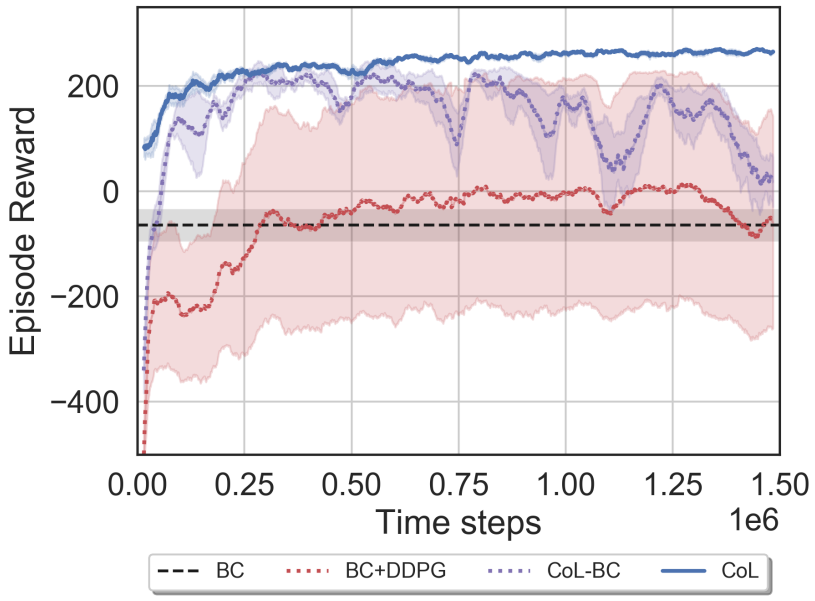


Figure 10.5: Effects of the combined loss in the Cycle-of-Learning. Results for 3 random seeds (bold line representing the mean and shaded area the standard error) showing component analysis in LunarLanderContinuous-v2 environment comparing complete Cycle-of-Learning (CoL), CoL without the expert behavior cloning loss (CoL-BC), and pre-training with BC followed by DDPG without combined loss (BC+DDPG). Reprinted from [9].

$$\begin{aligned}\mathcal{L}_{DDPG}(\theta_Q, \theta_\pi) =& \lambda_{Q_1}\mathcal{L}_{Q_1}(\theta_Q) + \lambda_A\mathcal{L}_A(\theta_\pi) \\ &+ \lambda_{L2Q}\mathcal{L}_{L2}(\theta_Q) + \lambda_{L2\pi}\mathcal{L}_{L2}(\theta_\pi).\end{aligned} \tag{10.7}$$

10.5.3.3 Effects of Human Experience Replay Sampling

To determine the effects of the experience replay buffer on performance we compare the standard CoL, which utilizes a fixed ratio buffer of samples comprising 25% expert data and 75% agent data, against an implementation with Prioritized Experience Replay (PER) [241], with a data buffer prioritized by the magnitude of each transition's temporal difference (TD) error, denoted as *CoL+PER*. The comparative performance of these implementations, for both the dense- (D) and sparse-reward (S) cases of the LunarLanderContinuous-v2 scenario, are shown in Figure 10.6. For

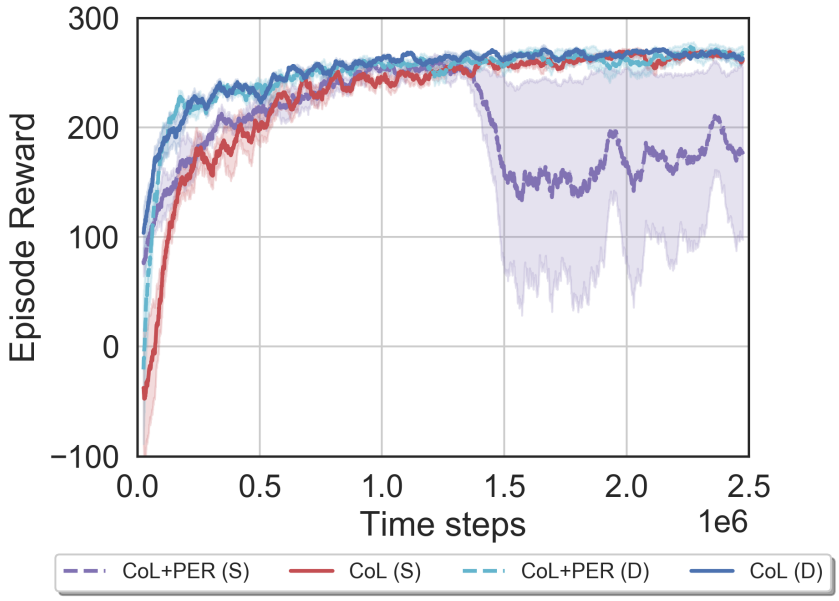


Figure 10.6: Effects of human experience replay sampling in the Cycle-of-Learning. Results for 3 random seeds (bold line representing the mean and shaded area the standard error) showing ablation study in LunarLanderContinuous-v2 environment, dense (D) and sparse (S) reward cases, comparing complete Cycle-of-Learning (CoL) trained with fixed ratio of expert and agent samples and complete Cycle-of-Learning using Prioritized Experience Replay (CoL+PER) with a variable ratio of expert and agent samples ranked based on their temporal difference error. Reprinted from [9].

the dense-reward condition, there is no significant difference in the learning performance between the fixed ratio and PER buffers. However, for the sparse-reward case of the *CoL+PER* implementation, the learning breaks down after approximately 1.3 million training steps, resulting in a significantly decreased performance thereafter. This result illustrates that the fixed sampling ratio for the replay buffer in the standard CoL is a more robust mechanism of incorporating experience data, particularly in sparse-reward environments, likely because it grounds performance to demonstrated human behavior throughout training.

10.6 Summary

In this work, we present a novel method for combining behavior cloning with reinforcement learning using an actor-critic architecture that implements a combined loss function and

a demonstration-based pre-training phase. We compare our approach against state-of-the-art baselines, including BC, DDPG, and DAPG, and demonstrate the superiority of our method in terms of learning speed, stability, and performance with respect to these baselines. This is shown in the OpenAI Gym LunarLanderContinuous-v2 and the high-fidelity Microsoft AirSim quadrotor simulation environments in both dense and sparse reward settings. This result is especially noticeable in the AirSim landing task (Figure 10.3), an environment designed to exhibit a high degree of stochasticity. The BC and DDPG baselines fail to converge to an effective and stable policy after five million training steps on the LunarLanderContinuous-v2 environment with dense reward and the modified version with a sparse reward signal. DAPG, although successful in both LunarLanderContinuous-v2 environments and the custom AirSim landing task, converges at a slower rate when compared to the proposed method and starts the training at a lower performance value after pre-training with demonstration data. Conversely, our method, CoL, is able to quickly achieve high performance without degradation, surpassing both behavior cloning and reinforcement learning algorithms alone, in both dense and sparse reward cases. Additionally, we demonstrate through separate analyses of several components of our architecture that pre-training, the use of a combined loss function, and a fixed ratio of human-generated experience are critical to the performance improvements. This component analysis also indicated that simply sequencing standard behavior cloning and reinforcement learning algorithms does not produce these gains and highlighted the importance of grounding the training to the demonstrated data by using a fixed ratio of expert and agent trajectories in the experience replay buffer.

Future work will investigate how to effectively integrate multiple forms of human feedback into an efficient human-in-the-loop RL system capable of rapidly adapting autonomous systems in dynamically changing environments. Actor-critic methods, such as the CoL method proposed in this paper, provide an interesting opportunity to integrate different human feedback modalities as additional learning signals at different stages of policy learning [1]. For example, existing works have shown the utility of leveraging human interventions [8, 233], and specifically learning a predictive model of what actions to ignore at every time step [242], which could be used to

improve the quality of the actor's policy. Deep reinforcement learning with human evaluative feedback has also been shown to quickly train policies across a variety of domains [232, 243] and can be a particularly useful approach when the human is unable to provide a demonstration of desired behavior but can articulate when desired behavior is achieved. Feedback of this type can be interpreted as a critique of an agent's current behavior relative to the human's expectation of desired behavior [243], thus making it conceptually similar to an advantage function which can be used to improve the quality of the critic. Further, the capability our approach provides, to transition from a limited number of human demonstrations to a baseline behavior cloning agent and subsequent improvement through reinforcement learning without significant losses in performance, is largely motivated by the goal of human-in-the-loop learning on physical systems. Thus our aim is to integrate this method onto such systems and demonstrate rapid, safe, and stable learning from limited human interaction.

11. CONCLUSIONS AND RECOMMENDATIONS

The following conclusions are made based on the results presented in this dissertation:

1. The Cycle-of-Learning is able to quickly achieve high performance without degradation, surpassing both behavior cloning and reinforcement learning algorithms alone, despite receiving only sparse rewards. Compared CoL to BC, DDPG, and DAPG, showing that CoL converges to better policy performance on the continuous environments studied. On LunarLanderContinuous-v2, this translates to 636% better when compared to BC, 71% better compared to DDPG, and 104% better than DAPG. Not only converges to better performance, but it starts with average performance higher than the baselines and, more important, behavior cloning. On the Microsoft AirSim environment, this translates to 161% initial higher performance when compared to DAPG, and 297% when compared to BC.

2. Component Analysis showed that the pre-training phase, each loss component, and the different forms of combining expert and agent samples in the experience replay are critical to the performance improvements. This component analysis also indicated that simply sequencing standard behavior cloning and reinforcement learning algorithms does not produce these gains.

3. Learning from demonstrations in combination with learning from interventions yields a more proficient policy based on less data, when compared to either approach in isolation. Averaging the results over all presented conditions and datasets, the task completion increased by 12.8% ($\pm$ 3.6% std. error) using 32.1% ($\pm$ 3.2% std. error) less human samples, which results in a CoL agent that overall has a task completion rate per sample 1.84 times higher than its counterparts. This supports the notion that the Cycle-of-Learning is a more data efficient approach to training via human inputs and further supports the notion that a combinatorial learning strategy inherently samples more data rich inputs from the human observer.

4. Human interaction modalities, or any expert prior knowledge or policy, should be exploited in all aspects of the Reinforcement Learning cycle for better results, and not only on the reward function.

5. Further, the case studies presented the application of inverse reinforcement learning and generative models to learn from human demonstrations, showing that an agent could learn how to perform a UAS landing task and surpass human mean performance after only 100 learning iterations.

Several recommendations are made based on the research in this dissertation:

1. Investigate the trade-offs between learning in a simulated environment and transferring to hardware compared to directly learning in hardware. High-fidelity simulation environments take great effort and time to be implemented and there are no theoretical guarantees that the policy will be transferred successfully.

    - How: As shown in this research with the Cycle-of-Learning and similar algorithms, as DAPG, if expert demonstrations are available, it is possible to integrate them to the reinforcement learning loop and learn directly in hardware. The time allocated to the development of the simulated environment can be shifted to collecting expert demonstrations.

2. Investigate the impact of neural network architecture size, mainly the number of hidden layers and neurons, on task performance when using machine learning algorithms, focusing on learning direct on hardware.

    - How: Advances in the fields of Neural Architecture Search (NAS) and Neural Network Compression can be leverage to find smaller architectures with similar accuracy that can increase inference speed and allow complex models to be deployed on hardware.

3. Develop new algorithms focusing on realistic, and consequently more complex, scenarios, even at the cost of reduced performance. Although simple environments are important to de-

velop and benchmark new algorithms, there is no theoretical guarantees that the performance will hold when they are applied to more realistic scenarios.

- How: When working with simulated environments, consider adding sensor noise, time synchronization, and high-fidelity dynamic models. Machine learning models are generally not robust to changes in the inputs distribution.

4. Consider the dynamic nature of human policies. This work assumed that the human policy remained fixed while training the learning agent and the human demonstrators were given practice time before performing the task. In reality, human performance can increase during training while the human learns to become better at performing the task or can degrade due to psychological factors.

    - How: Add a confidence metric to the behavior cloning loss based on expertise of the human expert or current psychological states.

5. Investigate how to extend Cycle-of-Learning to teach a learning agent multiple tasks, as opposed to a single task as presented in this dissertation. This is essential to create learning agents that are able to fulfil additional roles in real-time tasks.

    - How: Currently being developed by another graduate student in VSCL, Ritwik Bera, is a learning model that segments given demonstrations into differents options, which represent sub-tasks, and learns a option-conditioned policy.

REFERENCES

[1] N. R. Waytowich, V. G. Goecks, and V. J. Lawhern, "Cycle-of-learning for autonomous systems from human interaction," *CoRR*, vol. abs/1808.09572, 2018. [Online]. Available: http://arxiv.org/abs/1808.09572

[2] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[3] S. Levine, "UC Berkeley CS 294: Deep Reinforcement Learning, Lecture Notes," 2018, uRL: http://rll.berkeley.edu/deeprlcourse/. Last visited on 2019/09/01.

[4] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," *Computer Vision–ECCV 2014*, vol. 8689, pp. 818–833, 2014.

[5] V. G. Goecks and J. Valasek, *Deep Reinforcement Learning on Intelligent Motion Video Guidance for Unmanned Air System Ground Target Tracking*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2019-0137

[6] V. G. Goecks, P. B. Leal, T. White, J. Valasek, and D. J. Hartl, *Control of Morphing Wing Shapes with Deep Reinforcement Learning*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2018-2139

[7] V. G. Goecks, G. M. Gremillion, H. C. Lehman, and W. D. Nothwang, "Cyber-human approach for learning human intention and shape robotic behavior based on task demonstration," in *2018 International Joint Conference on Neural Networks (IJCNN)*, July 2018, pp. 1–7.

[8] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time," *CoRR*, vol. abs/1810.11545, 2018. [Online]. Available: http://arxiv.org/abs/1810.11545

[9] ——, "Integrating behavior cloning and reinforcement learning for improved performance in sparse reward environments," *ArXiv*, vol. abs/1910.04281, 2019.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Pearson Education, 2003.

[11] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.

[12] S. B. Kang and K. Ikeuchi, "Toward automatic robot instruction from perception-mapping human grasps to manipulator grasps," *IEEE transactions on robotics and automation*, vol. 13, no. 1, pp. 81–95, 1997.

[13] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Learning by watching: Extracting reusable task knowledge from visual observation of human performance," *IEEE transactions on robotics and automation*, vol. 10, no. 6, pp. 799–822, 1994.

[14] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[15] S. Schaal, "Learning from demonstration," in *Advances in neural information processing systems*, 1997, pp. 1040–1046.

[16] ——, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[17] C. G. Atkeson and S. Schaal, "Robot learning from demonstration," in *ICML*, vol. 97. Citeseer, 1997, pp. 12–20.

[18] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989, pp. 305–313.

[19] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," pp. 1–9, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[20] A. Giusti, J. Guzzi, D. C. Cire, F.-l. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots," vol. 3766, no. c, pp. 1–7, 2015.

[21] A. Nair, D. Chen, P. Agrawal, P. Isola, P. Abbeel, J. Malik, and S. Levine, "Combining

Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation," 2017. [Online]. Available: http://arxiv.org/abs/1703.02018

[22] F. Del Duchetto, A. Kucukyilmaz, L. Iocchi, and M. Hanheide, "Do not make the same mistakes again and again: Learning local recovery policies for navigation from human demonstrations," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4084–4091, 2018.

[23] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni, "Learning real manipulation tasks from virtual demonstrations using LSTM," 2016. [Online]. Available: http://arxiv.org/abs/1603.03833

[24] S. Hochreiter and J. Urgen Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 408–423, feb 2015.

[26] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 627–635. [Online]. Available: http://proceedings.mlr.press/v15/ross11a.html

[27] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive uav control in cluttered natural environments," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 1765–1772.

[28] C. Sammut and G. I. Webb, *Encyclopedia of machine learning*. Springer Science & Business Media, 2011.

[29] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," *arXiv preprint arXiv:1206.4617*, 2012.

[30] M. Johnson, N. Aghasadeghi, and T. Bretl, "Inverse optimal control for deterministic

continuous-time nonlinear systems," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 2906–2913.

[31] E. N. Sanchez and F. Ornelas-Tellez, *Discrete-time inverse optimal control for nonlinear systems*. CRC Press, 2017.

[32] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," *Proceedings of the Seventeenth International Conference on Machine Learning*, vol. 0, pp. 663–670, 2000. [Online]. Available: http://www-cs.stanford.edu/people/ang/papers/icml00-irl.pdf

[33] D. Ramachandran and E. Amir, "Bayesian inverse reinforcement learning." in *IJCAI*, vol. 7, 2007, pp. 2586–2591.

[34] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum Entropy Inverse Reinforcement Learning," *AAAI Conference on Artificial Intelligence*, pp. 1433–1438, 2008.

[35] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

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

[37] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz, "Keyframe-based Learning from Demonstration," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 343–355, 2012. [Online]. Available: http://link.springer.com/10.1007/s12369-012-0160-0

[38] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz, "Trajectories and keyframes for kinesthetic teaching," *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction - HRI '12*, p. 391, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2157689.2157815

[39] W. Saunders, G. Sastry, A. Stuhlmüller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," *CoRR*, vol. abs/1707.05173, 2017. [Online]. Available: http://arxiv.org/abs/1707.05173

[40] B. Hilleli and R. El-Yaniv, "Toward deep reinforcement learning without a simulator: An autonomous steering example," *AAAI Conference on Artificial Intelligence*, pp. 1471–1478,

2018. [Online]. Available: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16929

[41] M. Wigness and J. G. R. III, "On-line human intervention for robot behavior adaptation."

[42] B. Akgun, M. Cakmak, J. W. Yoo, and A. L. Thomaz, "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective," in *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. ACM, 2012, pp. 391–398.

[43] D. H. Grollman and O. C. Jenkins, "Dogged learning for robots," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 2483–2488.

[44] S. Javdani, S. S. Srinivasa, and J. A. Bagnell, "Shared autonomy via hindsight optimization," *Robotics science and systems: online proceedings*, vol. 2015, 2015.

[45] S. Reddy, S. Levine, and A. D. Dragan, "Shared autonomy via deep reinforcement learning," *CoRR*, vol. abs/1802.01744, 2018. [Online]. Available: http://arxiv.org/abs/1802.01744

[46] W. B. Knox and P. Stone, "Interactively Shaping Agents via Human Reinforcement: The TAMER Framework," in *International Conference on Knowledge Capture*, 2009.

[47] J. MacGlashan, M. K. Ho, R. T. Loftin, B. Peng, D. L. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback," *CoRR*, vol. abs/1701.06049, 2017. [Online]. Available: http://arxiv.org/abs/1701.06049

[48] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep tamer: Interactive agent shaping in high-dimensional state spaces," *AAAI Conference on Artificial Intelligence*, pp. 1545–1553, 2018. [Online]. Available: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16200

[49] A. L. Thomaz and C. Breazeal, "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," *Aaai*, vol. 6, pp. 1000–1005, 2006.

[50] A. Thomaz and C. Breazeal, "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance," in *AAAI Conference on Artificial Intelligence*, 2006.

[51] W. B. Knox, P. Stone, and C. Breazeal, "Training a robot via human feedback: A case study,"

in *International Conference on Social Robotics*, 2013.

[52] W. B. Knox and P. Stone, "Reinforcement learning from simultaneous human and MDP reward," in *International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[53] ——, "Framing reinforcement learning from human reward: Reward positivity, temporal discounting, episodicity, and performance," *Artificial Intelligence*, vol. 225, pp. 24–50, 2015.

[54] L. A. León, A. C. Tenorio, and E. F. Morales, "Human Interaction for Effective Reinforcement Learning," *Ke.Tu-Darmstadt.De*. [Online]. Available: http://www.ke.tu-darmstadt.de/events/PBRL-13/papers/04-Morales.pdf

[55] D. Arumugam, J. K. Lee, S. Saskin, and M. L. Littman, "Deep reinforcement learning from policy-dependent human feedback," *arXiv preprint arXiv:1902.04257*, 2019.

[56] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *arXiv*, 2017.

[57] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine, "End-to-end robotic reinforcement learning without reward engineering," *arXiv preprint arXiv:1904.07854*, 2019.

[58] B. Ibarz, J. Leike, T. Pohlen, G. Irving, S. Legg, and D. Amodei, "Reward learning from human preferences and demonstrations in atari," in *Advances in Neural Information Processing Systems*, 2018, pp. 8011–8023.

[59] J. Schulman, "Deep Reinforcement Learning - Machine Learning Summer Schools," Tech. Rep., 2016.

[60] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[61] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, 2000, pp. 1057–1063.

[62] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, 2002, pp. 267–274.

[63] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

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

[65] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." in *ICINCO (1)*, 2004, pp. 222–229.

[66] J. Schulman, S. Levine, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *Icml-2015*, p. 16, 2015. [Online]. Available: http://arxiv.org/abs/1502.0547

[67] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

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

[69] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[70] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. Eslami, M. Riedmiller *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint arXiv:1707.02286*, 2017.

[71] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," in *International Conference on Machine Learning*, 2016.

[72] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[73] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[74] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

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

[76] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[77] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[78] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," pp. 1–21, 2015. [Online]. Available: http://arxiv.org/abs/1511.05952

[79] Z. C. Lipton, J. Gao, L. Li, X. Li, F. Ahmed, and L. Deng, "Efficient exploration for dialogue policy learning with bbq networks & replay buffer spiking," *arXiv preprint arXiv:1608.05081*, vol. 3, 2016.

[80] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[81] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23*, J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, Eds. Curran Associates, Inc., 2010, pp. 2613–2621. [Online]. Available: http://papers.nips.cc/paper/3964-double-q-learning.pdf

[82] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[83] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume*

70. JMLR. org, 2017, pp. 449–458.

[84] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[85] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.

[86] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014.

[87] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, pp. 1–14, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[88] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[89] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

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

[91] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[92] S. S. Gu, T. Lillicrap, R. E. Turner, Z. Ghahramani, B. Schölkopf, and S. Levine, "Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning," in *Advances in neural information processing systems*, 2017, pp. 3846–3855.

[93] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-prop: Sample-efficient

policy gradient with an off-policy critic," *arXiv preprint arXiv:1611.02247*, 2016.

[94] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[95] R. J. Williams, "Simple Statistical Gradient-Following Algorithm for Connectionist Reinforcement Learning," *Machine Learning*, no. 8, pp. 229–256, 1992.

[96] I. Szita and A. Lörincz, "Learning Tetris using the noisy cross-entropy method." *Neural computation*, vol. 18, no. 12, pp. 2936–2941, 2006.

[97] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," *International Conference on Machine Learning*, pp. 745–750, 2007.

[98] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, "Model-based value estimation for efficient model-free reinforcement learning," *arXiv preprint arXiv:1803.00101*, 2018.

[99] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7559–7566.

[100] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[101] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, no. 4-5, pp. 667–682, 1999.

[102] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[103] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[104] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," *arXiv preprint arXiv:1401.4082*, 2014.

[105] C. M. Bishop, "Mixture density networks," Tech. Rep., 1994.

[106] B. A. Pearlmutter, "Learning state space trajectories in recurrent neural networks," *Neural Computation*, vol. 1, no. 2, pp. 263–269, 1989.

[107] A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland, "Finite state automata and simple recurrent networks," *Neural computation*, vol. 1, no. 3, pp. 372–381, 1989.

[108] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[109] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of IEEE international conference on evolutionary computation*. IEEE, 1996, pp. 312–317.

[110] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[111] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li *et al.*, "Imagination-augmented agents for deep reinforcement learning," in *Advances in neural information processing systems*, 2017, pp. 5690–5701.

[112] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," 2017. [Online]. Available: http://arxiv.org/abs/1705.05363

[113] J. Schmidhuber, "Curious model-building control systems," in *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*. IEEE, 1991, pp. 1458–1463.

[114] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," *arXiv preprint arXiv:1812.03201*, 2018.

[115] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[116] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew,

J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017, pp. 5048–5058.

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

[118] M. Večerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[119] T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. Van Hasselt, J. Quan, M. Večerík *et al.*, "Observe and look further: Achieving consistent performance on atari," *arXiv preprint arXiv:1805.11593*, 2018.

[120] G. V. Cruz Jr, Y. Du, and M. E. Taylor, "Jointly pre-training with supervised, autoencoder, and value losses for deep reinforcement learning," *arXiv preprint arXiv:1904.02206*, 2019.

[121] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6292–6299.

[122] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3309–3318.

[123] I. Durugkar, M. Hausknecht, A. Swaminathan, and P. MacAlpine, "Multi-preference actor critic," *arXiv preprint arXiv:1904.03295*, 2019.

[124] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," *arXiv preprint arXiv:1906.05841*, 2019.

[125] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *ACM Transactions on Graphics*

(TOG), vol. 37, no. 4, p. 143, 2018.

[126] R. Kaplan, C. Sauer, and A. Sosa, "Beating atari with natural language guided reinforcement learning," *arXiv preprint arXiv:1704.05539*, 2017.

[127] N. Waytowich, S. L. Barton, V. Lawhern, E. Stump, and G. Warnell, "Grounding natural language commands to starcraft ii game states for narration-guided reinforcement learning," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006. International Society for Optics and Photonics, 2019, p. 110060S.

[128] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[129] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard, "Way off-policy batch deep reinforcement learning of implicit human preferences in dialog," *arXiv preprint arXiv:1907.00456*, 2019.

[130] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters, "Active reward learning." in *Robotics: Science and systems*, 2014.

[131] P.-H. Su, M. Gasic, N. Mrksic, L. Rojas-Barahona, S. Ultes, D. Vandyke, T.-H. Wen, and S. Young, "On-line active reward learning for policy optimisation in spoken dialogue systems," *arXiv preprint arXiv:1605.07669*, 2016.

[132] M. Lopes, F. Melo, and L. Montesano, "Active learning for reward estimation in inverse reinforcement learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2009, pp. 31–46.

[133] B. Hilleli and R. El-Yaniv, "Deep learning of robotic tasks without a simulator using strong and weak human supervision," *arXiv preprint arXiv:1612.01086*, 2016.

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

[135] D.-A. Huang, D. Xu, Y. Zhu, A. Garg, S. Savarese, L. Fei-Fei, and J. C. Niebles, "Con-

tinuous relaxation of symbolic planner for one-shot imitation learning," *arXiv preprint arXiv:1908.06769*, 2019.

[136] J. Valasek, M. D. Tandale, and J. Rong, "A Reinforcement Learning - Adaptive Control Architecture for Morphing," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. April, pp. 174–195, 2005.

[137] J. Valasek, J. Doebbler, M. D. Tandale, and A. J. Meade, "Improved adaptive-reinforcement learning control for morphing unmanned air vehicles," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 38, no. 4, pp. 1014–1020, 2008.

[138] K. Kirkpatrick and J. Valasek, "Approximation of Agent Dynamics Using Reinforcement Learning," no. January, pp. 1–13, 2013.

[139] A. Lampton, J. Valasek, and M. Kumar, "Multiresolution state-space discretization for Q-Learning with pseudorandomized discretization," *Journal of Control Theory and Applications*, vol. 9, no. 3, pp. 431–439, 2011.

[140] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics :," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2015. [Online]. Available: http://ijr.sagepub.com/content/32/11/1238

[141] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," pp. 1–29, 2016. [Online]. Available: http://arxiv.org/abs/1606.06565

[142] S. Shahrdar, L. Menezes, and M. Nojoumian, "A survey on trust in autonomous systems," in *Intelligent Computing*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham: Springer International Publishing, 2019, pp. 368–386.

[143] D. Stormont, "Analyzing Human Trust of Autonomous Systems in Hazardous Environments," 2008.

[144] J. Carlson and R. R. Murphy, "An investigation of MML methods for fault diagnosis in mobile robots," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 1, 2004, pp. 180–186 vol.1.

[145] M. A. Abd, I. Gonzalez, M. Nojoumian, and E. D. Engeberg, "Trust , Satisfaction and

Frustration Measurements During Human-Robot Interaction," in *30th Florida Conference on Recent Advances in Robotics*, 2017, pp. 2–6.

[146] D. Howard and D. Dai, "Public perceptions of self-driving cars: The case of Berkeley, California," in *Transportation Research Board 93rd Annual Meeting*, vol. 14, no. 4502, 2014, pp. 1–16.

[147] A. Uggirala, A. K. Gramopadhye, B. J. Melloy, and J. E. Toler, "Measurement of trust in complex and dynamic systems using a quantitative approach," *International Journal of Industrial Ergonomics*, vol. 34, no. 3, pp. 175–186, 2004.

[148] T. Helldin, G. Falkman, M. Riveiro, and S. Davidsson, "Presenting system uncertainty in automotive UIs for supporting trust calibration in autonomous driving," in *Proceedings of the 5th international conference on automotive user interfaces and interactive vehicular applications*. ACM, 2013, pp. 210–217.

[149] L. Kunze, N. Hawes, T. Duckett, M. Hanheide, and T. Krajník, "Artificial intelligence for long-term robot autonomy: A survey," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4023–4030, Oct 2018.

[150] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54 – 71, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608019300231

[151] J. H. Block and L. W. Anderson, "Mastery learning," *Handbook on Teaching Educational Psychology*, 1974.

[152] C.-L. C. Kulik, J. A. Kulik, and R. L. Bangert-Drowns, "Effectiveness of mastery learning programs: A meta-analysis," *Review of educational research*, vol. 60, no. 2, pp. 265–299, 1990.

[153] S. A. Anderson, "Synthesis of research on mastery learning." 1994.

[154] D. Barber, *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[155] D. Ardila, A. P. Kiraly, S. Bharadwaj, B. Choi, J. J. Reicher, L. Peng, D. Tse, M. Etemadi, W. Ye, G. Corrado *et al.*, "End-to-end lung cancer screening with three-dimensional deep

learning on low-dose chest computed tomography," *Nature medicine*, vol. 25, no. 6, p. 954, 2019.

[156] M. R. Vargas, B. S. De Lima, and A. G. Evsukoff, "Deep learning for stock market prediction from financial news articles," in *2017 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. IEEE, 2017, pp. 60–65.

[157] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[158] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA:, 2015, vol. 25.

[159] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[160] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks*, vol. 2, no. 3, pp. 183–192, 1989.

[161] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[162] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[163] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *IJCNN International Joint Conference on Neural Networks*, 1989.

[164] N. E. Cotter, "The stone-weierstrass theorem and its application to neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 4, pp. 290–295, 1990.

[165] Y. Ito, "Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory," *Neural Networks*, vol. 4, no. 3, pp. 385–394, 1991.

[166] V. Y. Kreinovich, "Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem," *Neural networks*, vol. 4, no. 3, pp. 381–383, 1991.

[167] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[168] B. D. Ripley and N. Hjort, *Pattern recognition and neural networks*. Cambridge university press, 1996.

[169] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[170] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Univ Ca Stanford Electronics Labs, Tech. Rep., 1960.

[171] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab Inc Buffalo NY, Tech. Rep., 1961.

[172] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, Tech. Rep., 1985.

[173] A. Karpathy *et al.*, "Cs231n convolutional neural networks for visual recognition," *Neural networks*, vol. 1, 2016.

[174] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[175] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 2nd ed. Athena Scientific, 2000.

[176] W. B. Powell, "AI, OR and Control Theory: A Rosetta Stone for Stochastic Optimization," 2012.

[177] L. C. Baird III, "Advantage updating," WRIGHT LAB WRIGHT-PATTERSON AFB OH, Tech. Rep., 1993.

[178] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 38, no. 8, p. 716, 1952.

[179] ——, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684,

1957.

[180] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, 1989.

[181] C. Szepesvári, "Algorithms for Reinforcement Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[182] T. M. Mitchell, *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, 1980.

[183] K. Grill-Spector and R. Malach, "The Human Visual Cortex," *Annual Review of Neuroscience*, vol. 27, no. 1, pp. 649–677, 2004. [Online]. Available: http://www.annualreviews.org/doi/10.1146/annurev.neuro.27.070203.144220

[184] B. De Martino, D. Kumaran, B. Seymour, and R. J. Dolan, "Frames, biases, and rational decision-making in the human brain," *Science*, vol. 313, no. 5787, pp. 684–687, 2006.

[185] A. Y. Ng, "Shaping and policy search in reinforcement learning," *ProQuest Dissertations and Theses*, vol. 3105322, pp. 155—-155 p., 2003.

[186] X Development LLC, "Project Wing," 2018. [Online]. Available: https://x.company/wing/.Lastvisitedon2018/05/31

[187] Amazon Prime Air, "Determining Safe Access with a Best Equipped, Best-Served Model for Small Unmanned Aircraft Systems," 2018, https://www.amazon.com/primeair. Last visited on 2018/05/31.

[188] "The Drone Co. Latest Projects," 2018, https://thedrone.co/projects/. Last visited on 2018/05/31.

[189] "Unmanned systems roadmap 2007-2032," United States Office of the Secretary of Defense, Tech. Rep., 2007.

[190] J. Richard A. Best, "Intelligence, surveillance, and reconnaissance (isr) programs: Issues for congress," CRS Report for Congress, Tech. Rep., 2005.

[191] J. Valasek, K. Kirkpatrick, J. May, and J. Harris, "Intelligent motion video guidance for unmanned air system ground target surveillance," *Journal of Aerospace Information Systems*,

vol. 13, no. 1, pp. 10–26, 2016, doi: 10.2514/1.I010198.

[192] C. Noren, J. Valasek, V. G. Goecks, C. Rogers, and E. Bowden, *Flight Testing of Intelligent Motion Video Guidance for Unmanned Air System Ground Target Surveillance*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2018-1632

[193] C. Dunn and J. Valasek, "Unmanned Air System Search and Localization Guidance Using Reinforcement Learning," *Proceedings of the 2012 AIAA Infotech@Aerospace Conference*, 2012.

[194] J. Egbert and R. Beard, "Low-altitude road following using strap-down cameras on miniature air vehicles," in *Proceedings of 2007 American Control Conference*, 2007, pp. 353–358, iEEE, Piscataway, NJ, USA July 2007, doi: 10.1109/ACC.2007.4282767.

[195] J. Saunders and R. W. Beard, "Visual tracking in wind with field of view constraints," *International Journal of Micro Air Vehicles*, vol. 3, no. 3, pp. 169–182, 2011, doi: 10.1260/1756-8293.3.3.169.

[196] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv*, pp. 1–9, 2016.

[197] J. Valasek, Ed., *Morphing Aerospace Vehicles and Structures*. Chichester, UK: John Wiley & Sons Ltd., 2012.

[198] A. Lampton, A. Niksch, and J. Valasek, "Morphing Airfoils with Four Morphing Parameters," *AIAA Guidance, Navigation and Control Conference and Exhibit*, no. August, 2008. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2008-7282

[199] P. B. C. Leal, H. R. Stroud, and D. J. Hartl, "Design and fabrication of a shape memory-based bio-inspired morphing wing," 2017.

[200] M. Kumar, S. Chakravorty, and J. Valasek, "A Hierarchical Control Approach to Morphing Dynamics," *AIAA Infotech@Aerospace Conference*, no. April, 2009. [Online]. Available: http://arc.aiaa.org/doi/10.2514/6.2009-1830

[201] D. Lagoudas, *Shape Memory Alloys: Modeling and Engineering Applications*, 1st ed. New York: Springer, 2008.

[202] S. Swei and K. Cheung, "Mission Adaptive Digital Composite Aerostructure Technologies," *NASA Ames Research Center*, 2016. [Online]. Available: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20160012458.pdf

[203] A. F. R. C. NASA, "NASA Spanwise Adaptive Wing Concept," 2016. [Online]. Available: https://www.nasa.gov/feature/nasa-to-test-in-flight-folding-spanwise-adaptive-wing-to-enhance-aircraft-efficiency

[204] A. Lampton, A. Niksch, and J. Valasek, "Reinforcement Learning of a Morphing Airfoil-Policy and Discrete Learning Analysis," *Journal of Aerospace Computing, Information, and Communication*, vol. 7, no. 8, pp. 241–260, 2010. [Online]. Available: http://arc.aiaa.org/doi/10.2514/1.48057

[205] J. Valasek, K. Kirkpatrick, and A. Lampton, "Morphing Unmanned Air Vehicle Intelligent Shape and Flight Control," *Morphing Aerospace Vehicles and Structures*, no. April, pp. 55–86, 2012.

[206] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[207] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization," *International Conference on Learning Representations 2015*, pp. 1–15, 2015.

[208] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[209] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[210] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[211] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference*

on computer vision, 2015, pp. 1026–1034.

[212] C. Finn, S. Levine, and P. Abbeel, "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization," *ICML 2016*, vol. 48, no. 2000, 2016. [Online]. Available: http://arxiv.org/abs/1603.00448

[213] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," *NIPS*, no. NIPS, pp. 4565–4573, 2016. [Online]. Available: https://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning

[214] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 1–.

[215] E. T. Jaynes, "Information theory and statistical mechanics," *Phys. Rev.*, vol. 106, no. 4, pp. 620–630, May 1957.

[216] C. Finn, P. F. Christiano, P. Abbeel, and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models," *CoRR*, vol. abs/1611.03852, 2016.

[217] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[218] J. Fu, "Inverse RL," Aug. 2018. [Online]. Available: https://github.com/justinjfu/inverse_rl

[219] S. Ross, G. J. Gordon, and J. A. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," vol. 15, 2010. [Online]. Available: http://arxiv.org/abs/1011.0686

[220] L.-J. Lin, "Reinforcement Learning for Robots Using Neural Networks," Ph.D. dissertation, Pittsburgh, PA, USA, 1992.

[221] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *In Advances in Neural Information Processing Systems 12*, pp. 1057–1063, 1999.

[222] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills," vol. 37, no. 4, 2018. [Online]. Available: http://arxiv.org/abs/1804.02717{%}0Ahttp://dx.doi.org/10.1145/3197517.3201311

[223] R. Ramakrishnan, E. Kamar, D. Dey, J. A. Shah, and E. Horvitz, "Discovering blind spots in reinforcement learning," *CoRR*, vol. abs/1805.08966, 2018. [Online]. Available: http://arxiv.org/abs/1805.08966

[224] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, "Learning dexterous in-hand manipulation," *arXiv preprint arXiv:1808.00177*, 2018.

[225] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[226] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[227] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018.

[228] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 6292–6299.

[229] S. Kakade, "A natural policy gradient," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, ser. NIPS'01. Cambridge, MA, USA: MIT Press, 2001, pp. 1531–1538. [Online]. Available: http://dl.acm.org/citation.cfm?id=2980539.2980738

[230] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: The tamer

framework," in *Proceedings of the fifth international conference on Knowledge capture*. ACM, 2009, pp. 9–16.

[231] J. MacGlashan, M. K. Ho, R. Loftin, B. Peng, G. Wang, D. L. Roberts, M. E. Taylor, and M. L. Littman, "Interactive learning from policy-dependent human feedback," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2285–2294.

[232] G. Warnell, N. Waytowich, V. Lawhern, and P. Stone, "Deep tamer: Interactive agent shaping in high-dimensional state spaces," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[233] W. Saunders, G. Sastry, A. Stuhlmueller, and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 2067–2069.

[234] D. Moorhouse and R. Woodcock, "US Military Specification MIL–F–8785C," *US Department of Defense*, 1980.

[235] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," 2014.

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

[237] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[238] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[239] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Dapg for dexterous hand manipulation," https://github.com/aravindr93/hand_dapg, 2018.

[240] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by

exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[241] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[242] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 3562–3573. [Online]. Available: http://papers.nips.cc/paper/7615-learn-what-not-to-learn-action-elimination-with-deep-reinforcement-learning.pdf

[243] D. Arumugam, S. Karamcheti, N. Gopalan, L. L. S. Wong, and S. Tellex, "Accurately and efficiently interpreting human-robot instructions of varying granularities," *CoRR*, vol. abs/1704.06616, 2017. [Online]. Available: http://arxiv.org/abs/1704.06616