Multi-Task Reinforcement Learning: From Single-Agent to Multi-Agent Systems

Matthew L. Trang

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Thinh Doan, Chair
Ryan K. Williams
Andrea L'Afflitto

November 28, 2022
Blacksburg, Virginia

Keywords: Reinforcement Learning, Drones, Catastrophic Forgetting, Multi-Agent Reinforcement Learning, Multi-Task Reinforcement Learning

Copyright 2023, Matthew L. Trang

Multi-Task Reinforcement Learning: From Single-Agent to Multi-Agent Systems

Matthew L. Trang

(ABSTRACT)

Generalized collaborative drones are a technology that has many potential benefits. General purpose drones that can handle exploration, navigation, manipulation, and more without having to be reprogrammed would be an immense breakthrough for usability and adoption of the technology. The ability to develop these multi-task, multi-agent drone systems is limited by the lack of available training environments, as well as deficiencies of multi-task learning due to a phenomenon known as catastrophic forgetting. In this thesis, we present a set of simulation environments for exploring the abilities of multi-task drone systems and provide a platform for testing agents in incremental single-agent and multi-agent learning scenarios. The multi-task platform is an extension of an existing drone simulation environment written in Python using the PyBullet Physics Simulation Engine, with these environments incorporated. Using this platform, we present an analysis of Incremental Learning and detail the beneficial impacts of using the technique for multi-task learning, with respect to multi-task learning speed and catastrophic forgetting. Finally, we introduce a novel algorithm, Incremental Learning with Second-Order Approximation Regularization (IL-SOAR), to mitigate some of the effects of catastrophic forgetting in multi-task learning. We show the impact of this method and contrast the performance relative to a multi-agent multi-task approach using a centralized policy sharing algorithm.

Multi-Task Reinforcement Learning: From Single-Agent to Multi-Agent Systems

Matthew L. Trang

(GENERAL AUDIENCE ABSTRACT)

Machine Learning techniques allow drones to be trained to achieve tasks which are otherwise time-consuming or difficult. The goal of this thesis is to facilitate the work of creating these complex drone machine learning systems by exploring Reinforcement Learning (RL), a field of machine learning which involves learning the correct actions to take through experience. Currently, RL methods are effective in the design of drones which are able to solve one particular task. The next step in this technology is to develop RL systems which are able to handle generalization and perform well across multiple tasks. In this thesis, simulation environments for drones to learn complex tasks are created, and algorithms which are able to train drones in multiple hard tasks are developed and tested. We explore the benefits of using a specific multi-task training technique known as Incremental Learning. Additionally, we consider one of the prohibitive factors of multi-task machine learning-based solutions, the degradation problem of agent performance on previously learned tasks, known as catastrophic forgetting. We create an algorithm that aims to prevent the impact of forgetting when training drones sequentially on new tasks. We contrast this approach with a multi-agent solution, where multiple drones learn simultaneously across the tasks.

Dedication

To Mom and Dad.

Acknowledgments

Thank you to Dr. Thinh Doan for being a voice of reason throughout this process and for teaching me to be a better researcher and student. The impact of your support and guidance has been immeasurable.

Contents

List of Figures

List of Tables

List of Abbreviations

AI Artificial Intelligence

DNE Does Not Exist

IL Incremental Learning

IL-SOAR Incremental Learning with Second-Order Approximation Regularization

MAPPO Multi-Agent Proximal Policy Optimization

MARL Multi-Agent Reinforcement Learning

MTRL Multi-Task Reinforcement Learning

PPO Proximal Policy Optimization

RL Reinforcement Learning

UAV Unmanned Aerial Vehicle

Chapter 1

Introduction

Collaborative drone, or unmanned aerial vehicle (UAV), technology has been rising in importance over the past decade and has been used in a variety of domains. Examples of UAV use can be seen in their applications for wildfire monitoring [1, 2, 3, 4], agricultural monitoring [5, 6], autonomous mapping [7, 8], and traffic monitoring [9]. Drone and UAV technology has become ubiquitous, with new applications for the technology arising constantly. However, one limiting factor of UAV technologies is that the development of these autonomous systems is difficult, and with the rise in potential applications, an approach to these systems which can generalize to different tasks is growing increasingly important. Advances in Reinforcement Learning (RL) have shown promise in demonstrating viability of the method to generate solutions to these complex systems. Reinforcement Learning is a method to create Artificial Intelligence (AI) agents which are able to choose the optimal actions to take in an environment to maximize the accumulated reward throughout an experience. With RL, complex drone systems have been successfully trained to accomplish tasks such as safe coordinated path-planning and navigation [10, 11, 12], field coverage [13], and drone-based delivery [14]. These RL approaches are designed to do well in one singular task. In order to consider general purpose drones, we explore the field of Multi-Task Reinforcement Learning (MTRL), which aims to create an RL agent which can generalize and apply knowledge across different tasks. Approaches to Multi-Task Reinforcement Learning include algorithms such as MT-Opt [15] or IMPALA [16], learning techniques such as curriculum learning [17]

or incremental learning [18], and many other innovations, such as gradient surgery [19], or Hindsight Experience Replay [20].

While MTRL shows significant promise in creating general task drone agents, MTRL does have a few significant limitations. One of these downfalls is that MTRL agents can encounter a problem known as catastrophic forgetting [21]. When agents undergo training, the learning process causes agents to occasionally forget what to do in an old task or scenario. This means that to train an RL agent on multiple tasks we risk overwriting information for the previously learned tasks. In scenarios where multiple policies can be stored, one option to avoid this problem could be to train the agent on each environment and store different policies for each task. However, this approach does not scale well and would be impossible in memory-limited scenarios. Thus, we study a solution to the catastrophic forgetting problem in cases where the agent uses only a single policy that can learn a new task without forgetting previous tasks, i.e., a restricted variant of incremental learning. With catastrophic forgetting in mind, we investigate methods of multi-task learning which can mitigate this issue. Additionally, we analyze how these training methods can benefit learning speed in general. In this thesis, we focus on approaching multi-task learning in the domain of navigational drones.

1.1 Goals and Objectives

In this thesis, an attempt at solving the catastrophic forgetting problem is made through the use of an algorithm we refer to as Incremental Learning with Second-Order Approximation Regularization (IL-SOAR). Incremental Learning (IL) is the process of learning different tasks sequentially, while using information, generally the policy, from previous tasks to continue learning. The IL-SOAR method is based on the concept of using a second-order Taylor series expansion to represent the previous environments, allowing us to simulate testing our

performance in an approximation of the previous environment, similar to rehearsal methods [22].

This approximation is required as we aim to solve multi-task learning in a setting where we cannot access old environments after moving on and do not store training information from the previous environments in the form of trajectories or episodic buffers. We impose this restriction as it is generally difficult and costly to test over different environments, as it would require simulating all environments and would increase computation time and storage requirements. While our technique does increase the amount of computational storage needed to train an agent compared to a generic IL approach, we are able to restrict our method to require a constant amount of memory for learning new environments regardless of the number of additional tasks we learn. We test the effect of our technique in mitigating catastrophic forgetting when compared to generic reinforcement learning.

We also study a multi-agent training scenario where multiple agents can train simultaneously in each of the tasks in question. While a multi-agent solution to the catastrophic forgetting problem violates the restriction on environment access and memory storage we place on the single agent case, the information is still useful as an comparison for optimality of the single agent case. The multi-agent scenario is trained using a centralized approach, where each agent trains in one for some amount of time to update its policy, then all the policies are aggregated, averaged, and redistributed to the agents by some fully-omniscient controller. This approach to multi-agent RL is known as Policy-Sharing, or Parameter-Sharing MARL [23].

In this thesis, we also study the statistical benefits of incremental learning in terms of training speed, to see if training an agent using incremental learning is more efficient than training a brand-new agent from scratch. We consider the benefits and detriments of incremental learning in a scenario where environments are increasing gradually in difficulty. We hope to

corroborate other research which shows that incremental learning or curriculum learning is beneficial in multi-task scenarios [24].

Lastly, we present a simulation framework for the testing and development of multi-task and multi-agent reinforcement learning. The simulation framework is an extension of the work done previously by Panerati et al. [25], who developed a multi-agent reinforcement learning environment called `gym-pybullet-drones`. Changes were made to include additional testing environments and configurable reward and termination scheming through configuration files for rapid RL development. The set of custom environments includes both multi-agent RL tasks and single-agent RL tasks. The code for the updated simulation framework can be found at https://github.com/trangml/multi-task-pybullet-drones.

1.2 Applications

As stated earlier, reinforcement learning for drones can be applied to a multitude of tasks, and with additional research in the field, we can achieve drones which are applicable to multiple tasks at once. Here, we present two applications of drone technology that drones could assist in.

1.2.1 Remote monitoring of wildfires using drones

Wildfires can cause disastrous amounts of damage to society and have recently become more frequent [1]. To protect against wildfires, it is important to monitor the fronts on which the wildfires may be spreading. One option to do so is the use of autonomous drones, which are able to stay above the dangerous flames and keep track of the fire boundaries until a team of firefighters can respond. Fires are generally unpredictable, thus designing an autonomous

drone system that can take action to avoid damage from the fires is a particularly difficult task. With RL, robust monitoring technologies could be made, and with some form of communication, a swarm of drones could potentially monitor a vast area of the wildfire [3, 26].

1.2.2 Autonomous mapping of urban areas

There are multiple contexts where mapping is a valuable task for drones to conduct, such as in a military scenario, disaster relief, and search and rescue. A swarm of drones is a good solution to these types of scenarios as they can cover a larger area than a team of humans and also reduce the risk significantly. For example, prior work in autonomous mapping [7] has led to UAV systems which are able to fly and monitor buildings to determine heat loss in order to maximize energy savings. A collaborative system of drones would can increase the speed at which such mapping takes place [8].

1.3 Challenges

Applying multi-task reinforcement learning to the domain of drones is challenging as the domain has continuous states and actions in a three-dimensional space and has policies that lead to unrecoverable states. This makes it difficult to find an optimal stationary state distribution [25]. The problem of catastrophic forgetting is also difficult to solve due to the complex, interconnected nature of deep neural networks and the backpropagation algorithm [21]. Applying RL to multi-agent systems also presents a plethora of difficulties because the problems for each agent become only partially observable, as the actions the other drones may take are unknown. Overall, the multi-task drone learning problem is particularly difficult,

which is one reason why the incremental learning approach taken in this research may be particularly effective.

1.4 Contributions

In summary, for this project, the following contributions are presented

- Simulation Environment:
    1. Training pipeline and environments for drones on a variety of single-agent and multi-agent navigational tasks.
    2. Configurable reward and termination schema for rapidly testing new environments.
    3. Combined RGB and kinematic observation space as inputs to the agents neural networks.
- RL agent training techniques:
    1. Demonstrations of the statistical impact of using incremental learning on the training time across tasks with increasing difficulties.
    2. Analysis of the impact of training an agent with incremental learning and incremental learning with SOAR on catastrophic forgetting.
    3. Analysis of the effectiveness of multi-agent centralized learning on multi-task learning.

We now provide background for our investigation of the following research questions:

- What is the statistical effect of incremental learning on training time using PPO? Additionally, does training using incremental learning on PPO speed up overall learning time when the tasks are incremental in difficulty, i.e., each task covers the same goal, but becomes slightly harder than the previous?

- What are the impact of incremental learning and incremental learning with SOAR on catastrophic forgetting in multi-task scenarios? How do multi-agent centralized learning methods perform on multi-task scenarios?

Chapter 2

Background

2.1 Markov Decision Processes

The concepts of this research are dependent on the mathematical properties of stochastic environments which are governed by a Markov Decision Process (MDP). A MDP is a modeling framework that defines how the transitions of states occur within an environment that is being acted upon by an agent and what response the agent receives for each transition [27]. Many problems can be considered to fall under the frameworks of MDPs, which provides us with the following mathematical formulations to solve them.

2.1.1 Mathematical Formulation of MDPs

For Markov Decision Process problems in RL, we generally define the scenario as part of an infinite-horizon problem. We represent these with the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P}$ is the transition probability matrix that specifies the next state given the current state and action, r is the reward function that maps a reward value to the specific transition, and $\gamma \in (0, 1)$ is the discount factor for future rewards.

The MDP tuple is interacted with by a policy, π, which is the mapping from $\pi : \mathcal{S} \to \mathcal{A}$. More specifically, the policy represents the probability distribution $\pi(\cdot|s)$ over the set of actions $\mathcal{A}$. Given some policy, we can determine the value function, $V^{\pi}(s)$, as a representation of

the long-term reward of following the policy from some state s.

The value function measures how good a state is overall, by factoring the current state and action as well as a prediction of future rewards, generally done by computing a discounted return of steps in the future. This is expressed as,

$$V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi}[r(s,a)] + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}(\pi(a|s)) V^{\pi}(s') \tag{2.1}$$

or

$$V^{\pi}(s) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) | s_0 = s, a_k \sim \pi(\cdot|s_k), s_{k+1}, s_{k+1} \sim \mathcal{P}(\cdot|s_k, a_k)]. \tag{2.2}$$

Similarly, we can denote the Q-function, which is the state-action value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow R$ represented by $Q^{\pi}(s,a)$, as

$$Q^{\pi}(s_t, a_t) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r(s_k, a_k) | s_0 = s, a_0 = a]. \tag{2.3}$$

Equation 2.1 is also known as the Bellman equation for RL [28], and it represents a recursive dependency of the expected value of the current state, $V^{\pi}(s)$, on the expected values of the next possible states, $V^{\pi}(s')$. This allows us to use the Bellman equation as an iterative update to estimate the value function. We know that due to the Fixed-point theorem, the Bellman equation will eventually converge to a unique V^{π}, and that this convergence point is an optimal policy, which we denote as π^*. Thus, we have the following equations for the Bellman optimality equation for the value and Q-value functions.

$$V^*(s) = \max_{a \in \mathcal{A}} \mathbb{E}[r(s,a)] + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}(a) V^*(s'). \tag{2.4}$$

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}(a) \max_{a' \in \mathcal{A}} Q^*(s', a'). \tag{2.5}$$

2.2 Reinforcement Learning

Reinforcement Learning is a subset of Machine Learning where an agent must learn a set of behaviors by interacting in an unknown environment [29]. In RL, an agent in any given state uses its policy to choose an action to take based on the observed state and receives a reward from the environment for doing so. This action also causes the agent's state observation to change, as dictated by the environment dynamics. The reward tells the agent how "good" the action it performed was at the previous state. This process continues, with the agent repeatedly taking actions and receiving rewards, until the environment terminates, or the agent reaches a goal state. Reinforcement learning uses the output reward from the environment to update the agent's policy and progress toward the optimal policy in an environment. Essentially, reinforcement learning allows an agent with no prior information about the environment to learn a strategy about how to interact with the environment and maximize the accumulated rewards throughout its interactions [30, 31, 32]. In Figure 2.1, a simplified model of RL interaction is illustrated.

One of the key aspects of reinforcement learning is that RL problems follow the properties defined by an MDP. Crucially, this means that there is some optimal policy that can be achieved by leveraging Bellman's Equation, as described above. This is the foundation for many of the theorems and proofs regarding the optimality and convergence of RL, and is the basis for the value iteration and policy iteration approaches to solving RL problems [29]. This has led to the traditional RL field, which has typically consisted of tabular methods or linear function approximator to estimate the action-value function. This is expanded on and made more capable by Deep Reinforcement Learning.

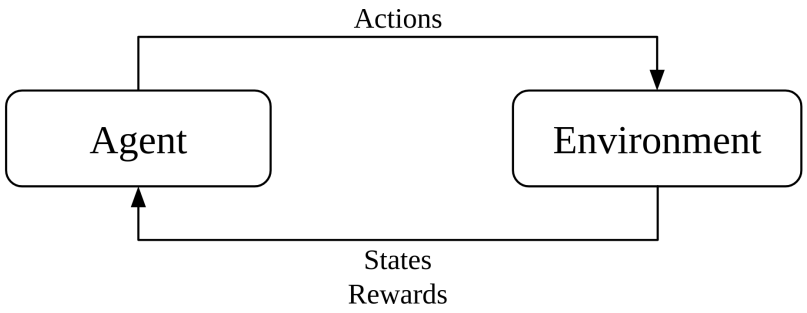


Figure 2.1: Simplified Reinforcement Learning interaction loop

2.2.1 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is an extension of RL which uses a deep neural network to do the function approximation for aspects of the RL problem which are difficult to model, such as the policy [33]. In many practical problems, most MDPs have high-dimensional state spaces which makes it difficult to solve using traditional RL. Most modern advancements in reinforcement learning are in DRL, as the usage of deep neural networks as function approximators has allowed RL algorithms to solve several major problems. One of the first critical breakthroughs of DRL was when the Deep Q-Network algorithm was used to solve Atari games [34]. Since then, deep reinforcement learning has been used to solve problems such as Chess, Go, StarCraft II, and many more [32, 35, 36, 37, 38]. Many different reinforcement learning algorithms exist, but for this project, we focus on policy gradient methods.

2.2.2 Policy Gradient

Policy gradient methods are one of the main approaches to achieving a policy with maximizes the expected return in an environment. Policy gradient methods refer to reinforcement learning algorithms that store and update a policy as opposed to storing and updating

values of a state [29]. These methods generally keep a model of the policy as a parameterized function with respect to $\theta, \pi_\theta(a|s)$. This policy is then updated by calculating a gradient with respect to the reward function and then using gradient ascent to update the parameters of the policy to approach an optimal stationary point. Some examples of Policy Gradient-based RL algorithms include REINFORCE [39], A2C [40], Deep Deterministic Policy Gradient (DDPG) [41], and Proximal Policy Optimization (PPO) [42]. A commonality behind many of these techniques is that they use estimates of the gradient of the policy, then use this value in stochastic gradient ascent or descent algorithms like Adam or AdaMax [43]. We know that by using these gradient methods, we can find the optimal policy by finding the stationary point via the gradient domination condition,

$$f(\pi) - f(\pi^*) \lesssim \frac{1}{1-\gamma}(\pi - \pi^*)^T \nabla f(\pi). \tag{2.6}$$

Thus, we can solve policy gradient methods using policy iteration to approach the optimal policy. Because π is parameterized by θ, we can represent our objective function $f(\pi)$ as $f(\theta)$.

Beginning with the equation for the objective function, which represents the function which we want to maximize, we have

$$f(\theta) = \mathbb{E}_t \left[\log \pi_\theta \left(a_t \mid s_t \right) \hat{A}_t \right], \tag{2.7}$$

which we can differentiate with respect to θ in order to get

$$\nabla f(\theta) = \mathbb{E}_t \left[\nabla_\theta \log \pi_\theta \left(a_t \mid s_t \right) \hat{A}_t \right]. \tag{2.8}$$

In this, $\hat{A}_t$ is an estimator of the advantage function at timestep t, where the advantage

function is defined as

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

Thus, to optimize the policy, we can use this gradient to update the policy parameters

$$\theta_{k+1} = \theta_k + \alpha \nabla_{\theta} f(\theta_k). \tag{2.10}$$

Policy gradient methods use this update to iteratively progress towards an optimal policy. We discuss REINFORCE and PPO, two policy gradient reinforcement learning algorithms.

2.2.3 REINFORCE

Here we discuss one of the earliest policy gradient methods, REINFORCE [39]. REINFORCE is also known as Monte-Carlo policy gradient, as it requires an agent to generate a full trajectory of an episode to then be optimized using the policy gradient theorem.

Algorithm 1: REINFORCE, Monte-Carlo Policy Gradient

Data: a differentiable policy parameterization $\pi_\theta(a|s)$, a learning rate $\alpha > 0$
Initialize θ arbitrarily
for *each episode* $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
  for $t = 1$ *to* $T - 1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    $\theta \leftarrow \theta + \alpha \gamma^t G \nabla_\theta \ln \pi_\theta(s_t, a_t)$
  end
end

The algorithm samples a trajectory based on the current policy, uses this data to perform an update on the gradient, then discards the collected trajectory. This results in an algorithm with poor sample efficiency. Additionally, as the trajectories the agent follows can change very rapidly with any update, the accumulated reward of each trajectory will have high variance, causing the algorithm to experience instability. These issues cause the algorithm

to be impractical for most use cases, and other modified policy gradient approaches which improve upon these deficiencies have become more common.

2.2.4 Proximal Policy Optimization

The Proximal Policy Optimization (PPO) algorithm is a model-free reinforcement learning algorithm that aims to solve some of the issues of REINFORCE [42]. PPO is a policy gradient method based on an algorithm called Trust Region Policy Optimization (TRPO) [44], where the goal is to update the policy in a restricted manner so as to remain close to a known policy. TRPO is not commonly used due to its computational requirements and dependency on the natural gradient, however, the concept of constraining policy updates with a KL divergence policy was proven effective. Thus, PPO simplifies the algorithm by using a clipping method to limit the amount it changes the policy at each step. Additionally, by using the advantage as the estimator for the return instead of the Q-function or value function, the agent can assess if an action was better or worse than expected, thus addressing some of the credit assignment issues other algorithms may face. This method also helps to reduce variance, as the advantage is always relative to a learned baseline in the Q-function. Lastly, to improve the sample efficiency, PPO can learn on a batch of data multiple times by using an importance sampling term to determine the weight that old samples can effect the new policy [45].

PPO can be formulated as follows, as described in Schulman's paper. We denote the probability ratio between old and new policies as, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. The restricted update of PPO aims to keep this probability ratio within a small interval in order to achieve stability. Thus, we have the following objective function for the policy of PPO

$$f^{\text{PPO}}(\theta) = \mathbb{E}\left[\min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a)\right)\right]. \tag{2.11}$$

Where $\hat{A}_\theta(s, a)$ is an estimator of the advantage function, and the function clip$(r(\theta), 1-\epsilon, 1+\epsilon)$ clips the input ratio between $1+\epsilon$ and $1-\epsilon$, where ϵ is some constant. The minimum operator makes it so that the smaller value of the original value and the clipped version is used to enforce controllable policy updates.

PPO is well known for ease of use and effectiveness in a variety of cases without having to perform excessive hyperparameter tuning. Generally, without performing hyperparameter tuning, PPO achieves better performance than other policy gradient algorithms such as DDPG and REINFORCE [42].

Since its creation, PPO has been one of the mainstay algorithms for generic single-agent reinforcement learning use cases, included in the UAV domain. For example, PPO has previously been used to train UAV attitude control [46]. In other works, PPO has been shown to be surprisingly effective when applied to multi-agent scenarios [23]. PPO-based algorithms have demonstrated robustness to environment nonstationarity common in multi-agent problems [47]. Thus, for the research we cover in the thesis, we use PPO as our main RL algorithm, for both the single-agent tests and as the training algorithm for the centralized multi-agent learning scenario.

2.3 Catastrophic Forgetting

Catastrophic forgetting is the phenomenon where an agent forgets how to solve a state that it was previously able to solve. The issue arises when an agent has to learn more information, and the new learning completely erases or invalidates previously learned information. The phenomenon has been a well-known issue in machine learning, being described as early as 1969 [48], however, the issue still plagues deep neural nets.

This catastrophic forgetting behavior of Deep RL agents is contrary to the way that the human mind works, and one of the main limitations preventing RL from achieving a generalized agent. For example, a human who has learned to ride a bicycle remembers how to ride, even after a significant amount of time and learning different tasks. Catastrophic forgetting makes it prohibitive for an RL agent to continuously learn new functions, as the agent would be unable to go back to solving the old functions without retraining once more.

There are many previous attempts to solve the problem of catastrophic forgetting in general machine learning. Approaches to solve the catastrophic forgetting problem with more advanced replay methods, new model mechanisms, and regularization factors like maximum entropy regularization or incremental iterative regularization are just a few of the previous attempts to solve the problem which can be found in the literature [49, 50, 51, 52, 53].

Other work has created benchmarks to measure catastrophic forgetting, as well as make comparisons between mechanisms such as regularization, ensemble methods, rehearsal, dual-memory, and sparse-coding as attempts to reduce the effects of catastrophic forgetting in deep neural networks for classification tasks. These methods were tested on incremental class learning for the supervised learning problem [54].

Similar work uses a separate memory system to represent continual task learning and reinforcement learning processes, as well as a pseudo-rehearsal system that generates representations of previous tasks using a deep generative network [22]. This allows the agent to continuously learn while not forgetting, but also without having increasing memory requirements for each task, having to store data generated from previous tasks, or revisiting prior tasks.

For reinforcement learning problems, the goal of solving catastrophic forgetting has been explored through training techniques such as elastic weight consolidation augmentation,

and training using algorithms designed for multi-task RL in experiments [55, 56]. These results have been shown to benefit training and reduce the impact of catastrophic forgetting. This approach is the most similar to our approach in the IL-SOAR algorithm, however, our augmentation technique does not use elastic weights, and instead uses environment approximations which we will describe the details of later.

2.4 Multi-Agent Reinforcement Learning

Humans demonstrate intelligence in a multitude of tasks and scenarios, not only due to our ability to learn by experience, but also by cooperation through sharing some form of information with others, by means of sharing methods of solving something, ideas about a problem, or instantaneous knowledge. The goal of Multi-Agent Reinforcement Learning (MARL) is to draw upon similar concepts to achieve performance greater than independent agents which do not cooperate during learning [57]. MARL systems refer to any RL problem where there are multiple agents which learn to take actions in a shared environment. The agents can interact with each other or can each have separate roles. An example of a MARL scenario could be a drone swarm which has to explore an area as a team. In such a scenario, all the drones must have the ability to learn to explore the areas. This means that for each drone, a policy could be learned. This policy could either be shared or learned individually. There are many different techniques to train multi-agent systems, such as having a central controller for all the agents, having each agent have individual policies, or hybrid methods, for example, techniques where agents each have a policy but share some policy weights [58]. Collaborative drone tasks can be considered multi-agent systems, as each drone would take an action.

Recent developments in MARL have included agents which can collaborate to play real-time

strategy games [37, 47, 59], autonomous driving and navigation [60, 61], autonomous drone field coverage [13], and card games [62]. MARL has some critical barriers, as the increase in agents leads to a much higher computational complexity, and the interactions between agents can cause the Markov properties of the environments they interact within to be invalidated. Furthermore, additional challenges such as partial observability and credit assignment are involved in MARL settings. Thus, steps have to be made to ensure that MARL algorithms can handle these challenges. Some well-known MARL algorithms include QTRAN [63] and QMIX [64].

Within the field of multi-agent RL, algorithms are generally classified based on of the training schemes they used, for example, centralized training methods and decentralized training methods. In centralized training methods, there exists some centralized point that is able to communicate with all of the individual agents, similar to the client/server model of coordination. The server uses information from each of the client agents, effectively causing the policies of any one agent to be impacted by all the agents [65]. In decentralized training methods, the policies of the agents are impacted by not all other agents, and instead, information is only shared between agents in local networks. In a decentralized learning scenario, there is no central server that communicates with every agent at once. However, there can be some shared communication between agents within their local networks [66]. In this project, we focus on centralized MARL settings.

2.4.1 Centralized Multi-Agent Reinforcement Learning

Centralized MARL is based on the ability to share learned policies or episodes among agents, which has been shown to contribute to expedited learning [67]. In centralized MARL, each individual client agent must communicate some information back to the central server. The

earliest approaches to multi-agent reinforcement learning explored three main ways of communication. The first was sharing instantaneous information such as the state, action, and reward at any specific timestep. Second, agents could share experience replay information or full trajectories of episodes. Lastly, agents could share learned policies. Since then, approaches to centralized MARL have explored variations of these communication techniques, with work done to share only specific parameters [68], policies [33], and other combinations showing merit. In our approach, we share policies of the agents, which results in the least amount of communication needed.

2.5 Multi-Task Reinforcement Learning

Multi-Task Reinforcement Learning (MTRL) refers to agents being able to generalize around different tasks. A generalized RL agent is an agent which is capable of performing multiple different things without having to be retrained or relearn weights [69]. For example, a well-trained general drone would be able to perform both a navigation task and a surveying task. One problem with the most commonly successful RL solutions of the past is that most RL results have been when a single agent learns to master a single task. To train an agent for a new task using traditional RL, oftentimes the training process is restarted from scratch, with each new task requiring a large amount of training time, computation, and data. This single-task limitation of RL is one of the main drawbacks of the field and falls behind the ability of humans to quickly learn and generalize to new tasks [70]. Improving the capabilities of RL agents to learn and adapt to multiple tasks is a key step in achieving generally useful AI, and general purpose drones [71].

MTRL algorithms have agents train in multiple tasks or environments and continuously learn new behaviors throughout training [69, 72, 73]. These different tasks could have different

goals but should have some similar characteristics or behaviors. For example, in other previous work, multi-task learning has included multi-armed bandits each with different reward probabilities, or mazes that all have different solution paths [74]. The intuition behind multi-task reinforcement learning is that within different simulation environments, there are shared aspects that an agent should be able to transfer over. For example, for a drone that is being trained to handle the task of exploring a room versus exploring the outdoors, the ability to fly should be retained between environments, similar to ideas used in previous robotic grasping work [75].

Some approaches to solving multi-task RL include using specifically designed algorithms, general learning techniques, or methods of imparting multi-task knowledge through parameterization or regularization. One algorithm for solving the problem is IMPALA [16], which uses importance weighting and decoupled acting and learning in order to perform well for generalization. However, in IMPALA, multiple tasks are learned in parallel, with separate learners allocated to each task. Similarly, MT-Opt is another learning method designed to handle multiple tasks, specifically for a robotics setting [15]. However, the algorithm relies on human examples, as well as storing data from continuous learning. This requires recording trajectory data for a specific task and carrying the trajectory data throughout learning, which can become restrictive and requires large data storage capabilities. Other approaches include representing the MDPs of different tasks within a hierarchical Bayesian infinite mixture model to represent shared characteristics of new environments [76]. Another learning paradigm for multi-task learning is Scheduled Auxiliary Control, which aims to use active scheduling to make agents explore [77].

Another method is to performing routing to determine which parameters are used for specific tasks [78], and performing gradient surgery to project gradients onto each other to detect conflicts [19]. Beyond these methods, training strategies can play a significant impact on

the ability of agents to learn multiple tasks [79]. These training strategies include transfer learning and curriculum learning, which we expand on later.

2.5.1 Mathematical representation of RL for a multi-task system

To represent the mathematical formulation for MTRL training, we utilize the descriptions presented by Zeng et al. [72]. In their paper, which focuses on decentralized policy gradient methods for multi-task learning, the authors present a formulation for the multi-task RL problem which holds regardless of decentralized learning or centralized learning.

We formalize the MDP with N tasks by using i to denote each task in N, with the tuple revising to $(\mathcal{S}_i, \mathcal{A}_i, \mathcal{P}_i, r_i, \gamma_i)$. Thus, for the multi-task system, the goal is to maximize the accumulated discounted rewards for each of the tasks as follows

$$\max_{\pi} V(\pi; \rho) \triangleq \sum_{i=1}^{N} V_i^{\pi}(\rho_i), \quad \rho = \begin{bmatrix} \rho_1 \\ \vdots \\ \rho_N \end{bmatrix}. \tag{2.12}$$

In this equation, ρ_i is an initial state distribution over $\mathcal{S}_i$.

In instances where each task has its own optimal policy, π_i^*, these policies would be those which maximize each respective value function, V_i^{π}. However, to store an optimal policy for each environment i would increase the storage requirement to scale linearly with N. We explore a system where each agent shares a policy that we define as π, and try to optimize the system over all agents to reach π^*, which is the most optimal over Equation 2.5.1. Thus, given this restriction, we can make our goal to find the single policy which optimizes the aggregate rewards by solving

$$\max_{\pi} f(\pi) \triangleq \sum_{i=1}^{N} f^i(\pi) \tag{2.13}$$

where $f^i(\pi)$ is the objective function for each task, and $f(\pi)$ is the overall objective of the MARL problem.

2.6 Transfer Learning

Having established the formulation for the problems and algorithms, we have set the background to discuss our overall learning techniques. Transfer learning for deep learning is when a model which was already trained is used to bootstrap training for another domain. The goal of transfer learning is to utilize the successfully trained network to perform knowledge transfer to the new domain and thus improve the learning performance [80, 81, 82]. Work done to study the process of multi-task reinforcement learning in humans have demonstrated that transfer learning is one method which humans use to generalize knowledge to solve new tasks [70]. Thus, we study transfer learning as it applies to this case.

The transfer learning process could include freezing some layers of the old model to train on top of it, using only limited steps, or any number of different techniques [82, 83]. Transfer learning could be used as one approach which addresses adapting to multiple tasks, however generic transfer learning usually does not have the goal of applying to the original distribution after retraining.

The literature for transfer learning is vast, but in this paper we mention a few of the advancements of transfer learning for reinforcement learning. Algorithmic approaches such as Actor-Mimic have attempted to solve the multitask learning problem by using transfer learning with agents learning in each task simultaneously with guidance [84].

In the drone domain, transfer learning has been explored in a number of other papers as well. A transfer learning based approach is used to reduce on-board computation of resource

constrained edge node drones in work [12]. Transfer learning has been used as part of simulation to real world transitions for quadcopters, with proven effectiveness in the real world [85].

2.7 Incremental Learning

We consider incremental learning, or curriculum learning, to be a multi-task reinforcement learning scenario where the tasks are learned sequentially to achieve a final difficult task. In this, a single agent trains and updates its policy on one task, then after a certain amount of training is given the next task to learn. This next task is a slightly more difficult version of the previous, and this process continues until we reach the final task [17].

Incremental learning can be considered a specific form of transfer learning, where the transferring environments conform to certain restrictions. Prior work in exploring the field has indicated empirically that when using an incremental curriculum, a significant boost in convergence speed is observed. Similar improvements are seen in generalization abilities [24]. Successful incremental learning could be considered as one of the key steps to achieving continual reinforcement learning, which is a requirement for agents which are able to learn and adapt to tasks in the real world [86].

Prior successes in area have shown that incremental reinforcement learning can be used to design multi-agent systems for toy gridworld environments [18]. By doing so, some of the effects of problems of multi-agent computational explosion, credit assignment, and partially observable environments can be reduced. However, despite this incremental learning continues to suffer from catastrophic forgetting.

Research has been done to explore avoiding catastrophic forgetting in incremental learning

through using an end-to-end training framework [87], and by incorporating concepts such as distillation loss [49]. Other work has been done to achieve incremental learning with regularization methods like dropout and Maximum Entropy Regularizer, which effectively induce a curriculum [52]. In these methods, learning is adapted on specific important nodes in order to control forgetting. However, none of these methods are complete solutions to the catastrophic forgetting problem in incremental learning.

2.7.1 Meta-Learning

The term Meta-Learning refers to overarching processes which allow an agent to learn to learn [88]. Throughout the years, the term has been applied to a variety of slightly different principles, but we use Meta-learning as defined by Schaul and Schmidhuber [89]. By their definition, meta-learning techniques are learning algorithms that change meta-parameters of a learning algorithm to impact performance gain. A meta-parameter is anything which can be altered within an algorithm, such as the learning rate, initial model weights, etc. The performance gain is considered as a relationship between the training experience and expected reward for that experience.

Meta-learning techniques are meant to address some of the deficiencies of learning techniques in learning different tasks, and are aimed to either improve learning speed or the quality of the learned policy. Both incremental learning and transfer learning could be considered "Meta-Learning" techniques as the methods aim to use previously acquired information to aid in learning [90]. Thus, training methods like incremental learning vary the meta-parameters of the initial model weights.

Prior work in meta-learning includes work done in inductive transfer, self-modification methods, and distributed learning algorithms, to name a few [89]. We focus on prior work for

meta-learning specifically for reinforcement learning using gradient methods. The process of applying gradient descent methods towards meta-learning has been explored in the context of learning using recurrent neural networks [91]. Meta-learning in terms of reinforcement learning has been discussed by Schmidhuber et al., and the technique has been shown to succeed even in conditions where traditional RL algorithms struggle, like non-Markovian environments [92]. In algorithmic approaches to meta-learning, prior work has explored the issue of backpropagating through adapting neural networks, which is difficult as the nodes may change throughout time [93]. Other meta-learning work explores using episodic recall to counter the catastrophic forgetting problem by using an LSTM [94]. Our approach is similar to these, as we aim to use previous information from old tasks to counter catastrophic forgetting, but without requiring the use of LSTMs or an adaptive neural network, but by instead changing initial model weights and parameter update functions.

2.8 Simulation Environment

Having discussed the algorithms behind the research, the next aspect which is important to discuss is the actual simulation environment chosen to conduct research. The environment selected was the `gym-pybullet-drones` environment developed for testing multi-agent quadcopter simulations [25]. The simulation environment is described as the first general-purpose multi-agent OpenAI Gym environment for quadcopters. The simulator that the environment used is the PyBullet simulator, which is a freely available open-source simulator which has been used in a large number of prior RL works [95].

PyBullet is designed to be used as a robotics simulator for research and education. It is a Python wrapper on the Bullet physics engine, a fast real-time simulation built in C++, and thus supports realistic collisions and aerodynamics at a level that other physics simulators

may lack. The relative performance of PyBullet compared to other popular simulators can be seen in prior work by Korber et al. [96]. The Python integration of PyBullet allows for easy interaction with reinforcement learning algorithms, which are primarily developed in Python as well. The `gym-pybullet-drones` environment came with simple single-agent reinforcement learning environments, such as a hover environment, as well as simple multi-agent environments, such as a leader-follower task.

This simulation environment was chosen over alternatives such as MuJoCo [97], AirSim [98], PEDRA [12, 99], CrazyS [100], Flightmare [101], etc., due to a number of reasons. Mainly, the benefits of the environment are that it provides realistic collisions, aerodynamics, and extensible dynamics, it has built-in OpenAI Gym style RL environments, it can be run in parallel with GPU, and it has pre-implemented StableBaselines [102] and RLlib workflows [103]. Additionally, creating simulation environments in `gym-pybullet-drones` is easy as objects can be loaded in using the Unified Robot Description Format (URDF), or Simulation Description Format (SDF), with configurable scaling, orientation, and collision information.

Chapter 3

Simulations

We hope to create simulation experiments to solve the problems stated earlier. These simulation environments are intended to be testing platforms for multi-task learning problems.

3.1 Simulation Setup

The simulation environments were set up as follows. The `gym-pybullet-drones` environment was updated with additional features and environments. For each environment, agents were trained using StableBaselines3, in either single agent or multi-agent approaches. We recorded these results on a local computer running Linux with NVIDIA Titan Xp GPUs. Here we provide additional information about the simulation environment.

3.1.1 Drone action spaces

The available action spaces for the drones in the `gym-pybullet-drones` simulation environment are as follows:

- RPM - Rotational speed of the motors in rotations per minute.
- DYN - Dynamics-based controls via the desired thrust and torque of the motors.
- PID - Proportional-Integral-Derivative Controller based control.

- VEL - Desired velocity vector input, given the X, Y, and Z directional unit vectors, and some magnitude.

The action choice chosen to represent the drone action space in these tests was RPM control, as the RL policy should be able to learn how to control the drone with any of the control schemes, and this method was tested in prior work. A further extension could be to explore the effectiveness of learning using a different control scheme. A simpler control scheme may show promise in creating more advanced policies.

3.1.2 Drone observation spaces

The drone observation spaces as packaged with the `gym-pybullet-drones` environment were limited to either RGB or kinematic information.

- RGB - RGB representation of the onboard drone camera, which provides a 64 x 48-pixel image view from the single forward-facing camera of the drone.
- KIN - Kinematic representation of the drone state, in terms of global X, Y, and Z positions, quaternion orientation, roll, pitch, yaw, velocity, and angular velocity all represented as scalar values as returned by the simulation environment.

As an extension in this project, the observation spaces were updated to include combined observation spaces, representing a drone with both image and kinematic information as input. Three new observation spaces, which combined RGB camera, camera segmentation, and camera depth observation spaces were created. Here we list the three new observation spaces.

- BOTH - RGB and Kinematic representations combined as a dictionary input.

- DEPTH - Camera Depth sensor and Kinematic representation of the drone state. The camera depth sensor returns 64 x 48 pixel-wise measurements of the distance to the first object the camera can detect.

- ALL - Combined RGB, Depth, Segmentation, and Kinematic representation for observation spaces. Segmentation data represents object segmentation information on a per-pixel basis for the objects that the camera can see.

The main content of the thesis will explore learning with a drone that uses the combined observation space which uses the RBG camera and kinematic information, indicated as BOTH above, although additional exploration was done to test the ability of the agents to solve the environments given only RGB or Kinematic observation spaces.

Including the camera information allows the drone to learn to navigate around obstacles, instead of following a path using the kinematic information, while the kinematic information helps the drone understand the reward states, which tend to be dependent on global position in the scenario. Examples of visual camera input for the agents can be seen in Figure 3.9.

3.1.3 Quadcopter model

The quadcopter used for the drone simulations was a Bitcraze Crazyflie 2.x nano. It was the default model provided by Panerati et al. [25] in `gym-pybullet-drones`, and was chosen due to its availability and usage throughout the field. The dynamics for the quadcopter are well modeled in other work [104, 105]. The simulation environment includes dynamics for complex quadcopter aerodynamic effects, such as drag, downwash, and the ground effect.

3.1.4 RL training

RL training was done using StableBaselines3 [102]. Initial tests were done on the effectiveness of few of the built-in algorithms, such as A2C, PPO, and DDPG. After verifying the relative effectiveness on the environments, PPO was chosen as the main RL algorithm to use for training agents in the project. CleanRL [106] algorithms were also tested but did not perform as effectively as the algorithms in StableBaselines3. For Multi-Agent training, RLlib [103] was tested, however, the choice was again made to use StableBaselines3 instead, as a centralized multi-agent PPO (MAPPO) algorithm was chosen as the primary multi-agent algorithm.

3.1.5 Regularized multi-task incremental learning with IL-SOAR

Here, we present the formulation of the IL-SOAR algorithm. The algorithm uses similar ideas to the Taylor Expansion Policy Optimization algorithm [107], with differences as our method does not aim to optimize over the same environment with off-policy data. The goal of our IL-SOAR algorithm is to maximize the cumulative reward across all the tasks that the agent will learn on and thus aim to solve the catastrophic forgetting problem. We provide the mathematical theory for the algorithm as follows.

Given N different tasks, each with different objective functions $f^1, f^2, \ldots, f^N$, we hope to achieve a single policy π which is parameterized by θ, $\pi_\theta(s, a)$ which will maximize the sum of accumulated rewards across all N tasks, as described in Equation 2.13.

In a scenario where we cannot return to an old task after training on it once, we are unable to directly evaluate the return of a new policy on the old policy. However, by using *Taylor series expansion*, if we keep the last policy trained on the old environment, denoted as π^{i-1}, and the last gradient of the objective function at the last step of training, ∇f^{i-1}, we can

make an approximation of how well a new policy, π, does on the old task.

$$f^{i-1}(\pi) \approx f^{i-1}(\pi^{i-1}) + \nabla f^{i-1}(\pi^{i-1})^T(\pi - \pi^{i-1}) + (\pi - \pi^{i-1})^T \frac{\nabla^2 f^{i-1}(\pi^{i-1})}{2}(\pi - \pi^{i-1}). \quad (3.1)$$

In this approximation, we have the objective of the previous environment used in the first approximation term $f^{i-1}(\pi^{i-1})$. However, in the policy gradient, we update the policy, π by differentiating f with respect to π. Thus, the approximation does not need to access the old environments, and can instead utilize the old gradients and policy as an approximation. To simplify Equation 3.1, we can approximate the Hessian Term, $\nabla^2 f_{i-1}(\theta)$ with some constant matrix, L, such as the identity matrix, since the gradient of f is Lipschitz continuous. Thus, we can rewrite the approximation as follows

$$f^{i-1}(\pi) \approx f^{i-1}(\pi^{i-1}) + \nabla f^{i-1}(\pi^{i-1})^T(\pi - \pi^{i-1}) + \frac{L}{2}||\pi - \pi^{i-1}||^2. \quad (3.2)$$

We denote the approximation of the previous task defined in Equation 3.2 as $g^i(\pi)$.

$$g^i(\pi) = f^i(\pi^i) + \nabla f^i(\pi^i)^T(\pi - \pi^i) + \frac{L}{2}||\pi - \pi^i||^2. \quad (3.3)$$

Now, we can generalize this approximation to N tasks. At environment i in N, we have the following problem:

$$\max_{\pi} f^i(\pi) + \lambda \frac{1}{i-1} \sum_{l=1}^{i-1} g^l(\pi). \quad (3.4)$$

To solve this using gradient descent, we need to get the gradient of $g^i(\pi)$, which is

$$\nabla g^i(\pi) = \nabla f^i(\pi^i) + L(\pi - \pi^i). \quad (3.5)$$

Thus, the gradient of the objective function when solving task i is

$$\nabla f^i(\pi) + \lambda \frac{1}{i-1} \sum_{l=1}^{i-1} \nabla g^l(\pi). \tag{3.6}$$

$$\nabla f^i(\pi) + \lambda \frac{1}{i-1} (\sum_{l=1}^{i-1} \nabla f^l(\pi^l) + L \sum_{l=1}^{i-1} (\pi - \pi^l)). \tag{3.7}$$

Thus, the policy update at step k for a policy gradient algorithm using second-order approximation is as follows

$$\pi^i_{k+1} = \pi^i_k + \alpha(\nabla f^i(\pi^i_k) + \lambda \frac{1}{i-1} \sum_{l=1}^{i-1} \nabla f^l(\pi^l) + \lambda \frac{1}{i-1} L \sum_{l=1}^{i-1} (\pi^i_k - \pi^l)). \tag{3.8}$$

3.1.6 Application of IL-SOAR to PPO

To use the IL-SOAR algorithm, the standard PPO algorithm from StableBaselines3 was updated to incorporate Equation 3.8, and the PPO algorithm was incrementally trained. During training, the first environment would effectively train with generic PPO, as both the gradient and the old policy terms are 0. Then, once the agent completed learning in the first environment, the training algorithm would store the policy and the final gradient of the objective function. These values are then passed into the training algorithm as parameters to be used during the PPO gradient update steps. Critically, for the PPO algorithm to continue to update the policy within known bounds, we add the two regularization terms to the gradient before we perform clipping on the gradient. The agent is trained with maximizing Equation 3.4 as its objective. Then, once finished, the returned policy can be averaged with the existing policy, as can the gradient. Thus, we achieve constant additional memory, while retaining a regularization term for previous tasks. One important difference between the IL and IL-SOAR methods is that for the results in the thesis, IL-SOAR did not use the previous

environments optimal policy as a starting point for the next environment, while IL did. In initial tests, when using IL-SOAR with the starting policy, agent performance was generally worse overall. This may be because the combination of the second-order term and beginning with a non-randomized policy was too restrictive to exploration. This is a topic for future exploration.

3.2 Environment Design

Here we discuss the environments designed for the project. We first focus on single-agent learning environments. Each different environment is referred to as an Aviary. For a brief overview of the environments, the first scenario was a single-agent environment where the drone attempted to land on a small box. The second scenario was an environment where the drone attempted to land on a small box, but the path to the box was obstructed by pillars. The third scenario was a rectangular hallway where the drone had to fly from one side of the hallway to the other side to be considered successful. This environment had 3 variations with multiple pillars within them. The fourth scenario was a variation of the third, but with 6 incremental versions, where pillar obstacles were added to the path of the drone. An additional set of scenarios were made by adding a wall on one side of the fourth scenario for all 6 variations. Finally, the last scenario was a room-like environment where the drone was tasked with flying from one end of the room to the other, and each room was filled with unique furniture objects in order to make navigation difficult. Here, we go into more detail on the environments, one by one.

3.2.1 Landing Aviary

The first environment created was a single-agent navigation task, where the agent had to fly from a starting point to a fixed landing zone, and then land on the box which represented the landing zone. The environment can be seen in Figure 3.1.

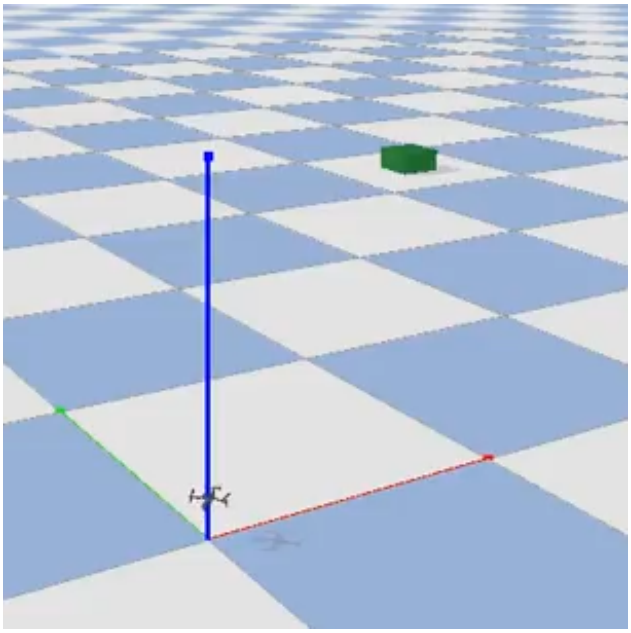


Figure 3.1: Landing Aviary with the drone at the starting point

3.2.2 Landing Obstacles Aviary

An extension to the Landing Aviary was created which contains the landing zone in the same area, but also has some obstructions in the way of the drone in the form of vertical pillars which stop the drone from being able to fly in the direct path to the landing zone. Figure 3.2 shows the environment.

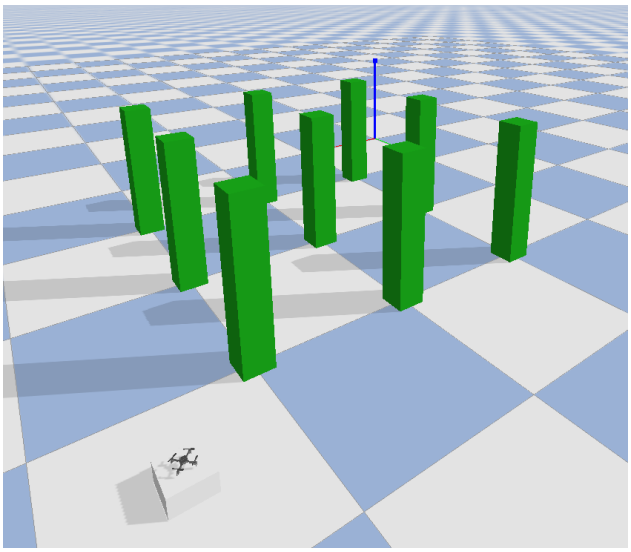


Figure 3.2: Landing Obstacles Aviary with the drone on the landing pad

3.2.3 Cross Obstacles Aviary

The Cross Obstacles environment tasked an agent to cross a long rectangular room and hover above a certain area to be considered as completing the task and solving the problem, mimicking an explorational task where the agent has to traverse some area. The environment can be seen in Figure 3.3 with the agent at the starting point.

For this environment, different task variations were designed corresponding to difficulty levels. The variations introduced vertical pillars which the drone had to avoid. Collision with these pillars, or with the walls of the room, would cause the simulation to terminate and the agent to receive a negative penalty. Figure 3.4 has the three different difficulty variations of the Cross Obstacles Aviary.

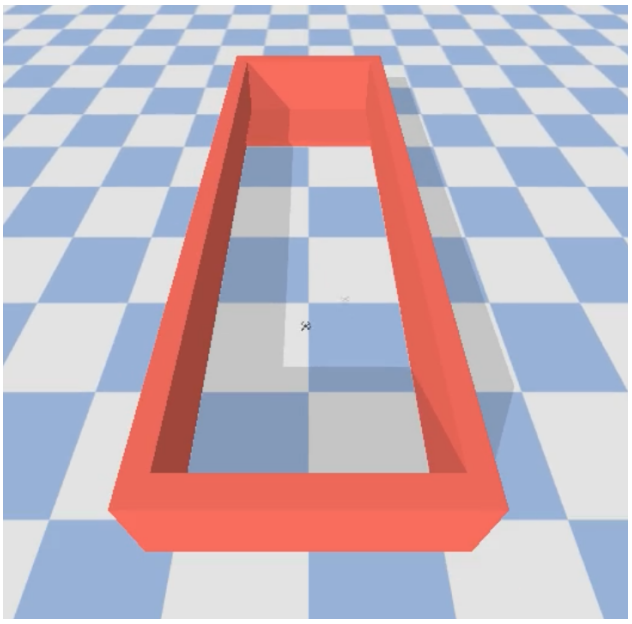
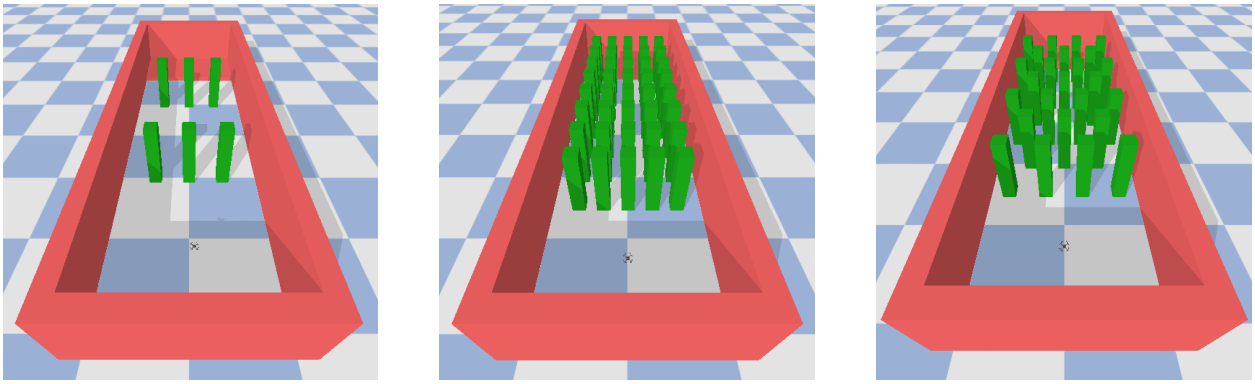


Figure 3.3: Cross Obstacles Aviary Difficulty 0



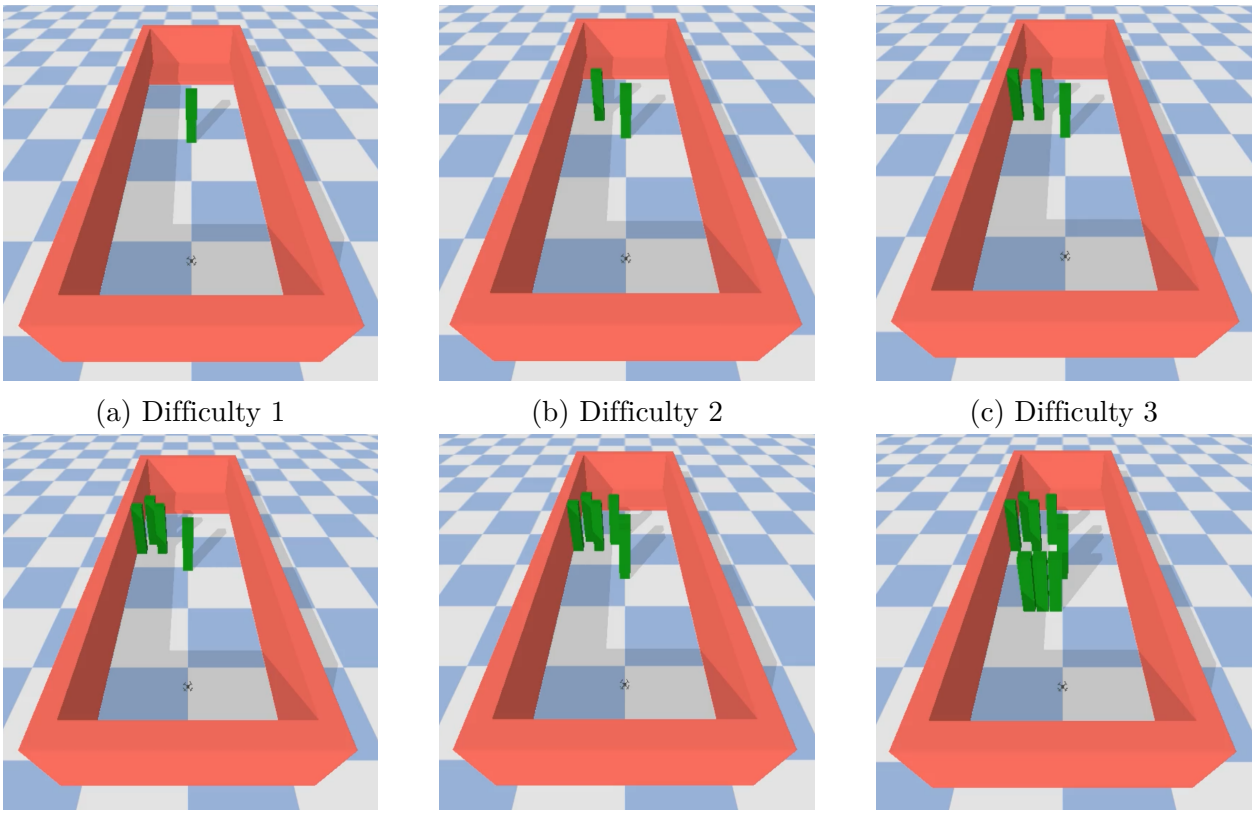
(a) Difficulty 1 (b) Difficulty 2 (c) Difficulty 3

Figure 3.4: Difficulty variations of the Cross Obstacles Aviary environments

3.2.4 Incremental Cross Obstacles Aviary

The incremental version of the cross obstacles aviary was a variation of the cross obstacles environment tested previously that was intended to be easier to solve for the RL agent. A single agent was first trained in an environment with no obstacles, then the flight path of the successful agent was used to define the next difficulty level. A single pillar was placed

as an obstacle in the direct flight path for the next difficulty environment. This process of training and then placing prohibitive obstacles was repeated 6 times, resulting in 6 different tasks with incrementally difficult obstacles, which can be seen in Figure 3.5.



(a) Difficulty 1 (b) Difficulty 2 (c) Difficulty 3

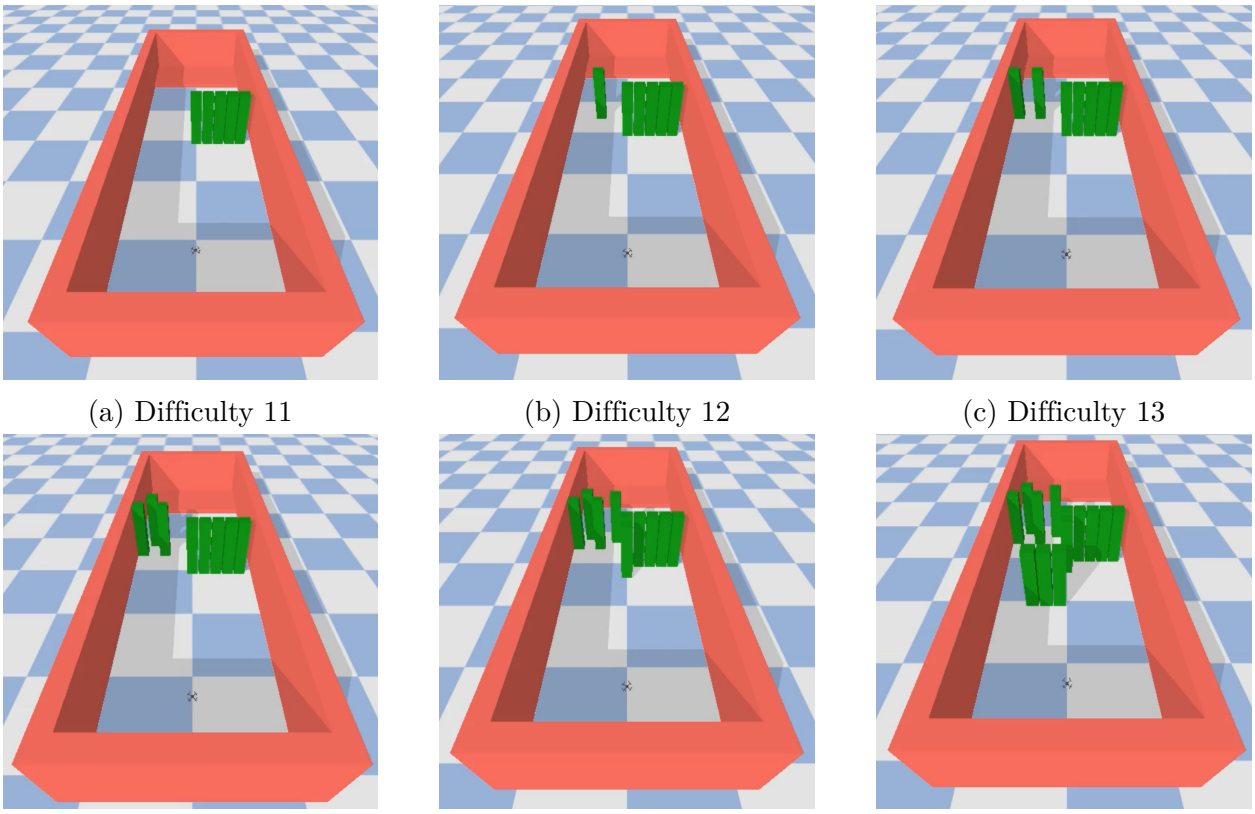
(d) Difficulty 4 (e) Difficulty 5 (f) Difficulty 6

Figure 3.5: Difficulty variations of the Incremental Cross Obstacles Aviary environments, each level has more pillar obstacles placed in the learned flight path

3.2.5 Harder Incremental Cross Obstacles Aviary

The incremental version of the Cross Obstacles Aviary was difficult for agents to solve, but if agents pursue the path along the right side of the room where there are no obstacles, each difficulty is equivalent. Thus, a harder version was created where a wall was placed on the right half of the room. Figure 3.6 shows this version, denoted the Harder Incremental

Cross Obstacles Aviary. The initial difficulty, Difficulty 0, was the environment shown in Figure 3.3. As these versions can be considered variations of the original Incremental Cross Obstacles Difficulty levels, we denote them as Difficulty 11-16 to distinguish between them.



(a) Difficulty 11 (b) Difficulty 12 (c) Difficulty 13

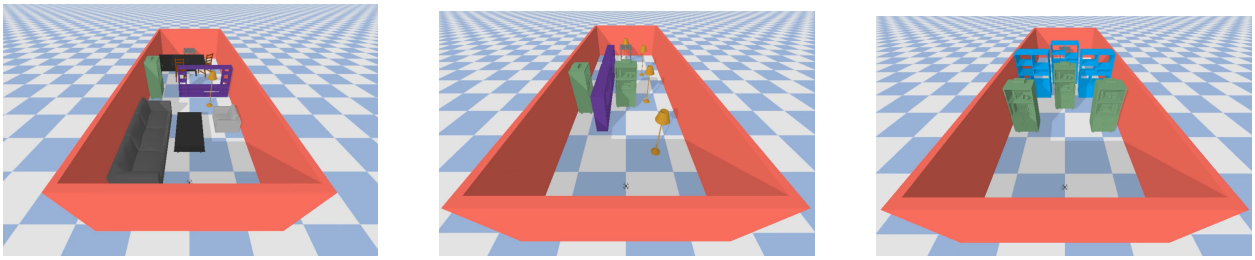
(d) Difficulty 14 (e) Difficulty 15 (f) Difficulty 16

Figure 3.6: Difficulty variations of the Harder Incremental Cross Obstacles Aviary environments, with the wall placed along the right side of the room

3.2.6 Room Aviary

The Room Aviary environments are extensions of the Cross Obstacles environments, but with more variation in the obstacles. In all three rooms, the objective remains to navigate the room and reach the other side, however, the agent is also strongly rewarded if it reaches the table at the end of the room. The three different room environments can be seen in

Figure 3.7.



(a) Room 0 (b) Room 1 (c) Room 2

Figure 3.7: Room Aviary environments, each room has a different arrangement of furniture and obstacles between the drone's starting point and the goal state

The goal of the agent is to reach the end of the room. In all three environments, the checkerboard table shown in Figure 3.8 is at the end. If the agent touches this table, the agent receives a large positive reward, and the environment is considered as solved.

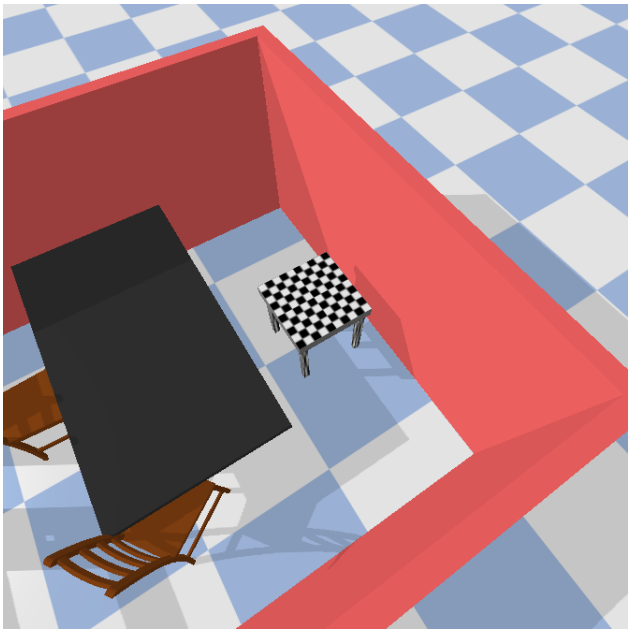
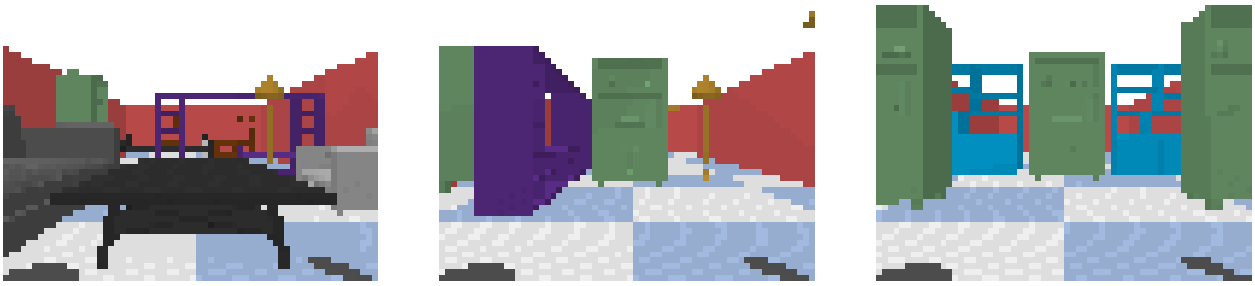


Figure 3.8: Room Aviary goal table, when the agents touch the table the episode ends and they receive a very large reward

Figure 3.9 shows the drone camera's perspective of the room at each of the starting frames. The camera detects RGB images which are 64x48 pixels.



(a) Room 0 (b) Room 1 (c) Room 2

Figure 3.9: Room Aviary tasks from drone camera view

Chapter 4

Results

Training was conducted for each of the environments, the results of which are presented here. First, simulations were done on the new environments to create benchmarks, and other initial observations and details are described. Multiple tests on each environment were conducted to first create a reward-shaping scheme that would allow the agent to approach the desired behavior. Next, simulations intended to explore the benefit of using incremental learning on training time for the PPO algorithm were conducted and explained. Experimentation for incremental learning was mainly for the Incremental Cross Obstacles Aviaries designed above, as well as the Room Aviary. After these simulation experiments, the results of incremental learning, IL-SOAR, and multi-agent learning on catastrophic forgetting are displayed as well.

4.1 Initial Results and Reward Shaping

4.1.1 Landing Aviary results

This simulation, while simple in design, turned out to be difficult for the agents to solve, and many observations were made when trying to train an agent to succeed in this task. The initial goal of the environment was to have the drone learn to fly and stop and land on the landing zone. Throughout testing, this was not achieved within 10 million timesteps using training with PPO, despite highly engineered reward schemes. Despite the agents not

fully solving the environment, important lessons were still learned, most importantly, the difficulties of creating a custom reward scheme for an environment.

Reward scheme exploitation

Reinforcement learning algorithms are well-known for reward function exploitation [108], which refers to unexpected behaviors that can occur as agents learn to maximize reward. For example, the initial reward function written for the landing task was a function that rewarded based on the inverse of the Euclidean distance function relating the location of the agent with respect to the landing zone. This led to the agent learning the behavior of moving along the floor on its propellers to reach the landing zone. The agent learned that it could maximize reward by leaning against the landing zone, seen in Figure 4.1, instead of flying and landing on top. This showed that it was necessary to write more complex reward and termination functions.

Figure 4.1: Landing Aviary exploiter, by leaning against the side of the landing zone, it is able to achieve high positive reward while not terminating the episode

Reward shaping creation

In order to create environments which agents would not exploit and would be able to solve, many different reward and termination schemes would need to be tested. In order to facilitate

this, the `gym-pybullet-drones` repository was extended with a system which allows the user to configure which reward and termination components are used by a reinforcement learning agent for training, instead of having the reward that an agent gets being strictly dictated by the environment. This was done using a YAML configuration file. By doing so, this allows for hyperparameter tuning algorithms to optimize over the weights of different rewards to create one which is most optimal for achieving some goal for a task. This system is available in the code repository linked earlier.

Using this, dozens of variations of the reward schemes were written and tested. Through these tests, it was found that using a combination of dense and sparse rewards was more effective. Training an agent to land with only the sparse reward of landing was an almost impossible task, as the algorithm has no guidance on all the prior actions to guide the agent to the landing zone. Thus, reward shaping needs to be done to assist the agent to reach its goal. The idea here is to create a dense-frequency reward that would give a smaller guiding value, but would be able to consistently guide agents. These functions include distance based reward functions, which reward positively for decreasing distance from the landing zone, or rewards functions that give a constant reward for staying level in the air. Each environment considered in these simulations will use a combination of sparse and dense rewards.

A few common reward schemes and termination conditions that were implemented were based on if an agent flew outside of configurable boundary conditions, if an agent reaches a certain roll or pitch angle, which are generally unsafe positions for quadcopters to be in, or if an agent reaches a certain area in the world. In the Incremental Cross Obstacles Aviary environments and forward, additional schemes were added based on if an agent collided with objects, such as termination if a drone hit a wall, or a large positive reward if a drone were to hit a goal object.

Impact of negative rewards

Through empirical testing, it was learned that negative dense rewards for a reinforcement learning problem were generally exploitable, as they can affect the agents will to survive. Effectively, when the agents were receiving negative dense rewards, such as a negative reward for being a certain distance from the goal, they would aim to terminate the simulation as early as possible to avoid accumulating the negative rewards. Even if the agent has the opportunity to achieve positive rewards later in the episode, due to how difficult the flying task is, agents would learn to reach this state. However, negative sparse rewards, such as punishments upon agent termination due to flying out of bounds were effective.

Learning to fly in the Landing Aviary

When given a reward scheme based on Euclidean distance as described earlier, the drones would consistently fly towards the ground and cause the episode to terminate. Thus, instead of taking the entire X, Y, and Z coordinates, the reward scheme was changed to take the X and Y coordinates only, in order to encourage the drone to stay in the air. Then, when the distance of the drone projected on the XY plane is within the bounds of the landing zone, an additional reward component based on the vertical distance to the landing zone is calculated to cause the agent to descend to the ground. This reward scheme was effective, and after more tuning, agents in the task could learn how to fly seen in Figure 4.2. However, despite learning to fly and hover near the landing zone, the drones were unable to learn how to descend and land.

One likely reason why the single-agent landing task was not successful was due to the difficulty of achieving fine-control tasks in reinforcement learning at the end of an episode. Once reaching a policy where the agent can navigate to the area above the landing zone, the agent

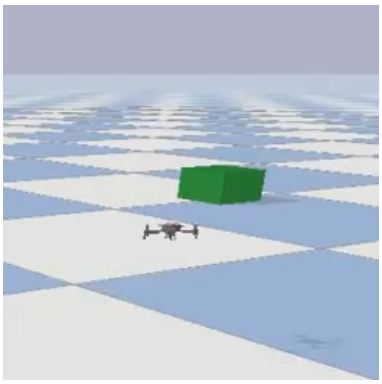


Figure 4.2: Drone flying in the Landing Aviary environment with a correct reward scheme

struggles to explore the states around that area and learn that it has to descend once reaching that location. The drone instead exhibits negative behaviors such as continuing past the landing zone, despite receiving a massive relative decrease in reward when leaving the high-value area of the landing zone. This can be attributed to the training algorithm used and the duration of training. With more training or techniques such as adaptive learning rates, the agent would theoretically be able to reach that optimal behavior. However, to provide a benchmark environment which is capable of being solved more easily, the scenario used was instead adapted. Solving the landing task remains an extension for future work.

4.1.2 Landing Obstacles Aviary results

The Landing Obstacles Aviary was a harder problem than the Landing Aviary due to the pillars, and the agents again failed to converge to a landing policy. Figure 4.3 shows a drone which hits the pillar, even after 10 million training timesteps. After testing that agents could not solve the original Landing Aviary task, tests for this environment are also considered as routes for future exploration.

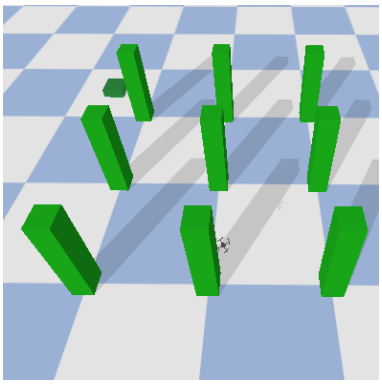


Figure 4.3: Drone flying in the Landing Obstacles Aviary, hitting a pillar

4.1.3 Cross Obstacles Aviary results

The new simulation environment created after the failure of the Landing Aviary Environment was a task that still involved navigation. Instead of also having the added difficulty of landing on the landing zone, the task was simplified. The objective for agents in these environments is to fly to the other side of the room and hover above a certain area.

For the three simpler versions of this task, referred to as Difficulty 0, 1, and 2 in Figures 3.3 and 3.4, the agents were able to solve it using PPO with 10 million training timesteps. However, for the hardest version, the agents were not able to fly to the other side. Additionally, the drones were found to exploit the environment by learning to flying into and rolling along the wall to solve the first three environments, instead of learning to follow optimal paths for each of them. Thus, for all future environments, a termination condition was added for anytime the drone collided with an object. To deal with the difficulty increase between the last two tasks being too steep, the Incremental Cross Obstacles Aviary was explored, creating our setting for properly exploring the effects of incremental learning.

4.2 Incremental Learning

For our tests in these environments, we explore the statistical effects of incremental learning on training time, and describe the benefits and drawbacks of the technique. We start with the Incremental Cross Obstacle Aviary.

4.2.1 Incremental Cross Obstacles Aviary results

Environment setup and reward scheme

For these tests, the agents were given reward functions that awarded 100 reward per timestep that the agent was in the goal area on the other side of the room, indicated by the horizontal red area at the ends in Figures 3.3 and 3.5. In this area, agents were allowed to collide with other objects, thus, as long as the agent reached the end of the room, it could stay on the red goal area and accumulate reward until the episode ended due to timeout. We consider a successful agent in this environment any agent who achieves greater than 1000 reward i.e., the goal in this scenario was to reach the goal state and survive for 10 timesteps.

Single seed results

An initial test on the impact of incremental learning on training was conducted where the number of training timesteps needed for the agent to achieve over 1000 reward on the environment was documented. Two techniques were compared, an incremental training technique, and generic RL with no pretraining or transfer learning involved. In the method that tested incremental learning, each difficulty began training by starting with the optimal policy of the previous difficulty. For both methods, agents were trained for up to 10 million timesteps. In the table, DNE entries indicate that the agent did not solve the problem within the 10

million timestep training frame.

	Training Method	
Difficulty	Incremental	Generic
0	412	412
1	258	592
2	150	450
3	120	890
4	14	DNE
5	2380	610
6	DNE	350

Table 4.1: Timesteps (in thousands) to get above 1K reward during training for Incremental Cross Obstacles Aviary

Analysis of single seed results

In this experiment, it was found that in general, incremental learning did decrease the number of training steps needed for each difficulty. We see that in difficulties 1-4, the number of training steps is less than that of the no-pretraining number. This is because the drone uses its existing knowledge to bootstrap the initial policy, then only makes finer changes as needed, instead of completely training a new policy that needs to learn how to fly.

However, in difficulties 5 and 6, the changes to the environment are more significant, with multiple obstacles being added to the environment at once. In these instances, the environment model seems different enough from the state distributions from environments 1-4 that the agent takes significantly longer to learn in environment 5 and fails to learn to solve the task in environment 6. To measure the overlap between the environments, it is possible to use the Fisher information matrices, a method used by Kirkpatrick et al. in prior work exploring overcoming catastrophic forgetting [109], however is beyond the scope of this project.

Cumulative single seed results

Now, if the number of steps needed to train the previous difficulty is included in the incremental training numbers, then the results become as follows

	Training Method	
Difficulty	Incremental	Generic
0	412	412
1	670	592
2	820	450
3	940	890
4	954	DNE
5	3334	610
6	DNE	350

Table 4.2: Cumulative timesteps (in thousands) to get above 1K reward during training for Incremental Cross Obstacles Aviary

Analysis of cumulative single seed results

Effectively, when considering the cumulative timesteps, the incremental method is detrimental to training for this environment. Because the incremental method continuously learns from the policy which followed the path the obstacles had been designed for, it does not explore along the right side of the room where there are no obstacles. A different agent trained from scratch on the obstacle environments would have a random policy to begin with, and thus would be more likely to follow the optimal path on the right side, albeit generally slower than the incremental approach.

These experimental results show that strong statistical benefits exist to incremental learning in terms of training time when the difference between environments is relatively small. If the time needed to train an agent on the previous task is inconsequential, for example, if the

drone has already trained for the previous task, then incremental learning is able to achieve optimal policies with less training time.

Testing across random seeds

After demonstrating that the agents were unable to achieve learning at some points, simulation experiments to see the generalization across multiple different random seeds were conducted to investigate if the effect was due to random exploration, or innate to the environment and algorithm. In Table 4.3, the average number of training timesteps across 5 different random seeds was recorded. The same incremental and no-pretraining test conducted in 4.1 was repeated for each.

	Training Method	
Difficulty	Incremental	Generic
0	442.4	442.4
1	102	587.2
2	33.6	2519.6
3	101.2	1652.4
4	8.8	3044
5	92	558
6	126.4	708

Table 4.3: Timesteps (in thousands) to get above 1K reward averaged over multiple random seeds during training for Incremental Cross Obstacles Aviary

Analysis of random seeds results

From this data, we can see that over multiple trials, the benefits of using incremental learning becomes more pronounced, as it can be seen that the agents which are not trained from an incremental version take millions of timesteps to solve environments 2, 3, and 4. Throughout the training, these agents tended to get stuck in local maxima areas. Interestingly, the

agents trained directly on difficulties 5 and 6 were able to learn faster than the three middle environments, demonstrating the volatility of the algorithm.

4.2.2 Harder Incremental Cross Obstacles Aviary results

In the previous results, one flaw is that for each tasks, if an agent learns to just follow the path along the right side, it would be able to solve the problem easily. Thus, to understand the full benefits of incremental learning in a scenario where an agent training from scratch would not necessarily have an advantage, we use the Harder Incremental Cross Obstacles Aviary defined in Section 3.2.5.

Environment setup and reward scheme

We conducted the same simulations done in the previous environment on these environments, with the same reward shaping, timesteps, and hyperparameters. As with the Incremental Cross Obstacles Aviary, any collisions of the drone with any obstacle or the walls outside of the goal area would result in the termination of the episode. Again, we consider success in this environment any agent with is able to reach above 1000 reward.

Testing across random seeds

The main experiment here is again to determine the amount of training timesteps needed by the algorithms to achieve success in training an agent. The results of training in the environment over several different random seeds can be seen in Table 4.4. The maximum amount of training that agents could perform was 10 million timesteps, and any agent which did not succeed in that time is denoted with a DNE (Does Not Exist).

	Training Method	
Difficulty	Incremental	Generic
0	442.4	442.4
11	156.5	654.4
12	73	1129.3
13	46.5	1558
14	34.5	1166
15	229.5	DNE
16	DNE	DNE

Table 4.4: Timesteps (in thousands) to get above 1K reward averaged over multiple random seeds during training for Harder Incremental Cross Obstacles Aviary

Analysis of random seeds results

These results show that incremental learning does improve training time in this case as well, further demonstrating the benefits of the technique. For this problem, when using incremental learning, the agent solves the Difficulty 15 environment within 10 million timesteps, which it is unable to do in the non-pretraining case. Thus, incremental learning can improve the ability of agents to solve hard tasks, that agents trained from scratch would have difficulties solving.

4.2.3 Room Aviary results

Additional work was done to explore the results of training an agent on the Room Aviary Environments, where each one is approximately equally difficult, and the optimal flight path of one environment is not the same as flight path for another environment. This forces the agents to combine the flight path information with visual reasoning to make more active choices of which flight paths to follow.

Environment setup and reward scheme

For the Room Aviary environments, the reward scheme was changed to reward more strongly for consistent stable flight and survival, as opposed to reaching the goal area, which was the main focus of tasks prior. The agents received dense rewards for maintaining level flight, avoiding collisions, and decreasing the distance between themselves and the goal area. The agent receives a high sparse reward for reaching the landing table at the end of the room, as described in Figure 3.8.

Training curves - generic Reinforcement Learning

To understand the impact that incremental learning makes on training, we examine the training curves for the three different algorithms. Figure 4.4 demonstrates the training results for agents in each of the three rooms using only generic RL with no pretraining or transfer learning.

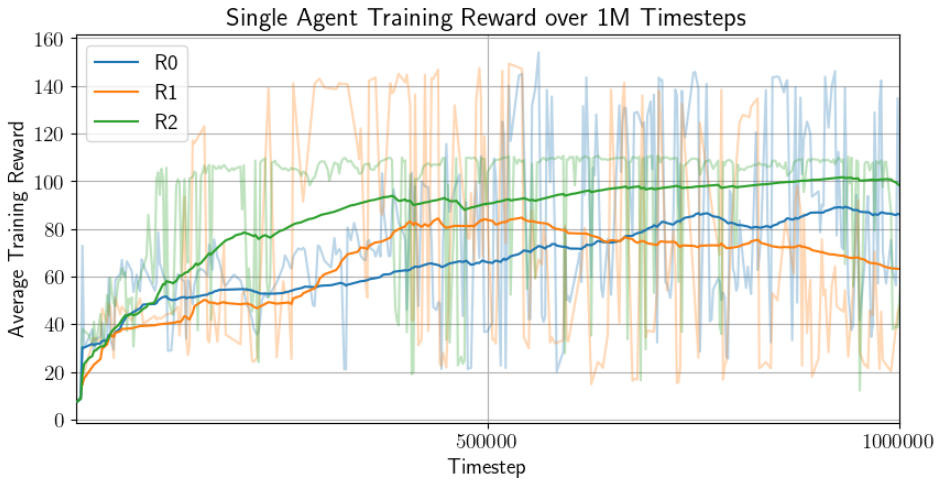


Figure 4.4: Training reward plot for generic single-agent learning in the three different Room Aviary environments

Training curves - Incremental Learning

Figure 4.5 shows training using incremental learning. Because each of the environments is effectively equivalent in difficulty, the order of incremental training is arbitrary. Thus, the order was chosen as Room 2 as the first environment, Room 0 as the second environment, and Room 1 as the last environment, following the maximum to least reward from Figure 4.4. The training curves of the agents can be seen in Figure 4.5. Each agent is incremental from the optimal policy of the agent before it.

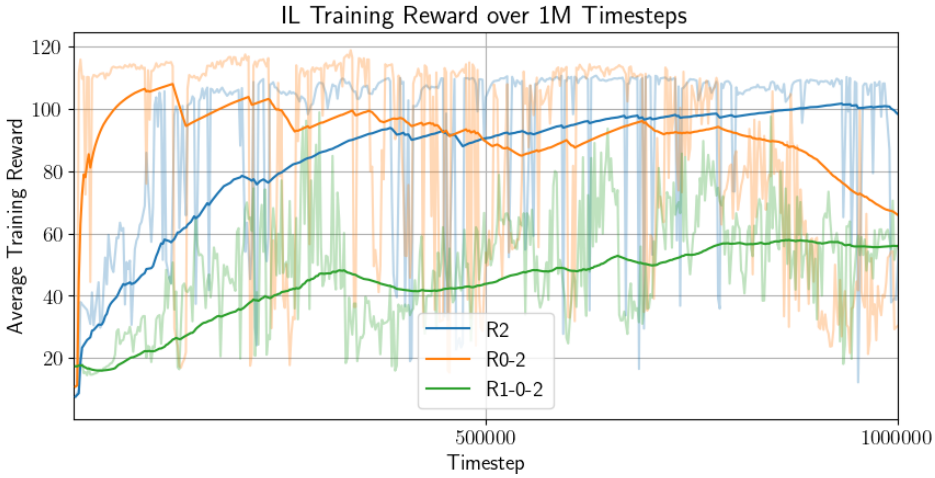


Figure 4.5: Training reward plot for Incremental Learning in the three different Room Aviary environments

Training curves - Incremental Learning with Second-Order Approximation Regularization

Finally, Figure 4.6 displays the training curves for agents trained using the IL-SOAR method described in Section 3.1.6. The agents are trained in the same order as the previous incremental training test.

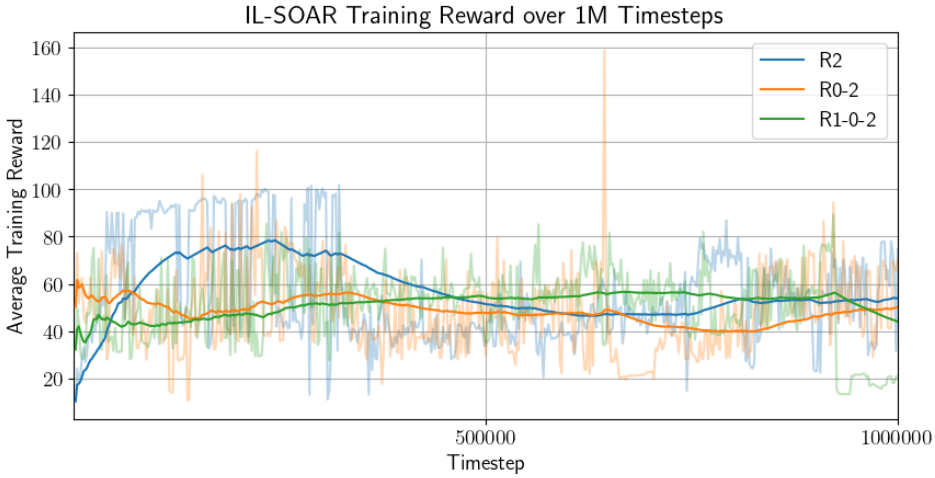


Figure 4.6: Training reward plot for Incremental Learning with SOAR in the three different Room Aviary environments.

Training curves - Multi-Agent Proximal Policy Optimization

In Figure 4.7, the training curves of agents using multi-agent PPO are shown. The agents were trained on all tasks simultaneously, with each agent learning in their individual environment for 5000 steps, then each policy is collected and averaged together by a central server. This policy is then redistributed to each agent for continued training, until a total of 1 million training timesteps is completed.

Analysis of training curves

Overall, training only on the specific task is the most effective in achieving high reward for one specific application. The peak achieved reward for agents in Figure 4.4 is higher than any other achieved reward throughout the other training methods. The incremental learning method is faster at achieving high reward for newly trained environments, as can be seen by the orange curve indicating good performance on R0 immediately from R2. However, the incremental learning method does not reach as optimal of a solution for the problems as

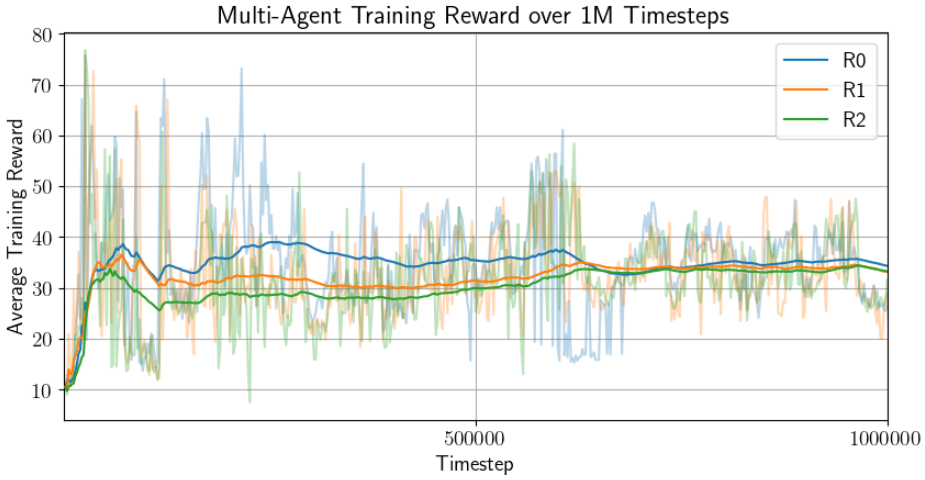


Figure 4.7: Training reward plot for multi-agent learning in the three different Room Aviary environments. Agents were trained with a Policy Sharing MAPPO approach.

that of the generic RL with no pretraining approach. For the IL-SOAR algorithm, while the average training curves appear lower in general throughout training, the peak performance for the algorithm are able to reach similar levels, and at times exceed that of IL. The final agent of the IL process, R1-0-2, is able to reach a final reward of around 50, while the IL-SOAR method shows a peak of around 70 before the training ends. In a later section, we discuss the actual evaluation reward of these trained agents.

4.3 Multi-Task Catastrophic Forgetting Exploration

Here we explore the other main question of the thesis, the impact of training methods on the catastrophic forgetting problem when training multi-task agents. To explore whether catastrophic forgetting has occurred, we first train on the tasks, in the single-agent case by using IL or IL-SOAR, and in the multi-agent case by using MAPPO. After completing training, we evaluate the final trained agent in the tasks it has learned on and denote its performance. We compare these results to performance of a single-agent which trained only

in that task. The first environment we tested the effects of catastrophic forgetting in was the Harder Incremental Cross Obstacles Aviary.

4.3.1 Harder Incremental Cross Obstacles Aviary catastrophic forgetting results

We begin our exploration of the catastrophic forgetting problem with insight into the Harder Incremental Cross Obstacles Aviary.

Experimental setup and reward scheme

Previously, training time to reach an optimal policy was tested. Instead, for this test, agents are trained for a fixed number of timesteps, which for this test was 1 million, and the value recorded was the reward that the resulting policy achieves on an evaluation environment. For the incremental learning case, every difficulty level was given the opportunity to train for 1 million timesteps, and the starting policy for learning was the best policy which was achieved by the previous task.

In this experiment, in order to more clearly see the forgetting effect, we change the reward function from what was discussed previously in Section 4.2.2. The new reward scheme for the environments scales the reward achieved in reaching the goal area of the environment to be significantly less, so the agents to focus less on achieving the goal and more on overall survival in the scenario.

Evaluation results

In Table 4.5, the reward accumulated for each of the different training methods was considered. The first 7 rows are the agent that had been trained only in one environment for 1 million timesteps, with no pretraining or transfer learning. The last 6 rows are the agents which were trained incrementally. These agents were each trained for 1 million timesteps, but from the starting point of the most optimal policy for the previous environment.

	Difficulty							
Training Method	0	1	2	3	4	5	6	Average
Trained on 0	447.82	82.5	46.23	3.08	1.77	22.02	1.15	86.36
Trained on 11	398.36	467.82	45.69	46.71	45.73	44.76	33.79	154.69
Trained on 12	462.82	465.82	463.82	44.7	45.71	45.7	19.51	221.15
Trained on 13	108.22	122.31	197.99	265.69	51.67	50.68	20.5	116.72
Trained on 14	258.7	337.61	169.8	220.24	338.46	49.73	25.29	199.97
Trained on 15	45.59	42.56	42.56	42.56	42.56	42.56	32.51	41.55
Trained on 16	40.6	32.53	34.54	32.52	32.52	36.55	42.59	35.97
Incremental to 11	197.14	472.82	45.66	45.65	47.93	45.87	24.86	125.70
Incremental to 12	161.95	148.52	220.23	105.21	44.69	49.75	23.73	107.72
Incremental to 13	44.91	472.82	260.64	463.82	37.77	55.72	24.88	194.36
Incremental to 14	55.84	126.35	475.82	172.84	152.34	49.65	21.54	150.62
Incremental to 15	25.61	221.39	206.24	245.62	41.76	44.77	25.6	115.85
Incremental to 16	25.64	26.64	27.65	26.64	26.64	28.67	28.69	27.22

Table 4.5: Reward accumulated after training for 1M timesteps for the Harder Incremental Cross Obstacle Tasks. Results are shown with agents trained directly on the environment and incremental agents up to a certain environment

Analysis

It can be seen that the regular incremental training did lead to catastrophic forgetting. The 1st Incremental Learning-based agent was only able to achieve 197.14 reward on the easiest 0-difficulty environment, as opposed to the 447 reward it initially received when trained in that environment. The agent would learn to maximize the reward in the environment it was

learning on without retaining information about the policy which was successful previously. The best performing agent overall was the agent which was only trained on difficulty 2, which was able to use a flight path that was close to optimal for all three of the first environments.

If the agents trained in the harder environments used only kinematic information, a policy trained in the higher level of difficulty should be able to achieve the same reward in a lower difficulty. In practice, this is not true, as the agents learn to rely on a combination of vision and kinematic information during training. Thus, when agents trained on higher-level environments are tested on the previous versions, they tend to perform much worse. This can be seen in the Incremental to 3 and Incremental to 4 agents.

Thus, the downsides of incremental learning with respect to catastrophic forgetting have been denoted. However, in this environment, the difficulty of the simulation increases, so a policy that can solve the hardest difficulty would theoretically be able to do equally as well on the easiest difficulty. We address this dependency in the next set of tests.

4.3.2 Room Aviary catastrophic forgetting results

To further test these results, we explore the effects of catastrophic forgetting in the Room Aviary.

Experiment setup and reward scheme

In this test, we keep the same reward scheme as defined earlier in Section 4.2.3. We again test each training method after training for 1 million timesteps in each environment, and compare to the results of a single agent trained for 1 million timesteps only in one environment.

Single agent evaluation results

Table 4.6 shows the reward achieved by the agents in each respective environment. Again, incremental agents were trained on environment Difficulty 2, then Difficulty 0, then Difficulty 1.

	Room			
Training Method	0	1	2	Average
Trained on 0	**104.91**	26.85	26.04	52.60
Trained on 1	12.06	**93.8**	19.26	41.71
Trained on 2	10.79	33.76	**110.7**	51.75
IL	53.8	59.35	33.07	48.74
IL-SOAR	68.02	77.05	60.60	**68.57**

Table 4.6: Reward accumulated after training for 1M timesteps in a Single-Agent setting

As evident in the table, the agents which trained directly on the environment with no pre-training achieve the most optimal policy for that environment. This is as expected, and parallels the results from Table 4.5. The performance of these agents on the two tasks they were not trained on was very poor and was below 35 reward for all of them. We can see that the Incremental Learning Method, which was trained in order on Rooms 2, 0, and 1, was able to achieve an average reward across the three tasks of 48.74, which is lower than two of the averages for the single agents, and its performance throughout the tasks is significantly lower compared to the agent which was optimal for the task. This shows that there is some catastrophic forgetting of the agent. The Incremental Learning with SOAR agent was able to achieve the highest average score across the three tasks, and higher individual scores in each of the three rooms. This shows that the incremental learning with SOAR method is effective in reducing the effects of catastrophic forgetting.

Multi-agent evaluation results

Table 4.7 shows the reward achieved by the agents in each respective environment. Lastly,

	Room			
Training Method	0	1	2	Average
Trained on 0	**104.91**	26.85	26.04	52.60
Trained on 1	12.06	**93.8**	19.26	41.71
Trained on 2	10.79	33.76	**110.7**	51.75
Multi-Agent PPO	65.85	60.14	54.86	**60.29**

Table 4.7: Reward accumulated after training for 1M timesteps in a Multi-Agent setting

the Multi-Agent PPO results are also better than incremental learning, with the multi-agent shared policy demonstrating higher reward on each task compared to regular Incremental Learning, but a lower overall average than IL-SOAR. One aspect of the multi-agent training results is that the agents seemed to be trending upwards at the end, indicating that they likely would have continued to improve if training for longer than 1 million timesteps.

Chapter 5

Conclusions

We have shown that for UAV navigational tasks using PPO, Incremental Learning shows significant benefits in improving training time for learning multiple tasks. We implemented simulation environments that demonstrated the effectiveness in a simulation environment that has incremental difficulties. We also demonstrated that incremental training can positively impact agents' abilities to solve hard tasks, which an agent training only on the task might not be able to solve. However, we have also demonstrated that incremental learning can be detrimental in instances where the new environment differs significantly from the original task, or if the agent which is being transferred is already in a local minimum.

In our exploration of catastrophic forgetting, we present an algorithm that demonstrates promising results in mitigating some of the negative effects of incremental learning on performance in old tasks. This algorithm, which we call Incremental Learning with Second Order Approximation Regularization or IL-SOAR, demonstrated the highest average results when tested across a set of tasks, even when compared to a Multi-Agent RL approach which was able to simultaneously explore all environments. However, the effectiveness of this technique depends heavily on the scaling at which the Second-Order Approximation term is applied, and the value for the scaling term will vary heavily depending on a simulation-to-simulation basis. Additionally, further improvements could be made the IL-SOAR algorithm to explore alternatives to the approximation for the second-order Hessian Term. In our thesis, the Hessian term is approximated simply with an identity matrix, however, other approximations

might function better. For example, the approximation could be a matrix that varies based on the importance of features, or something similar. More work needs to be done to establish how accurate of an approximation the IL-SOAR method provides, through comparing it to different multi-task learning algorithms like IMPALA. Lastly, the Incremental Learning with SOAR algorithm has yet to be tested on tasks in a different domain, which could be an avenue for discovering deficiencies of the algorithm. For our context, the rewards for the agents in each different task were the same scale. However, this might not be true for different tasks in general. This imbalance may lead to certain tasks impacting the overall policy more than another task. Thus, an improvement to this algorithm would account for this difference, perhaps by using PopArt normalization [73].

Finally, while the theoretical and simulated data for the effectiveness of Incremental Learning and Incremental Learning with SOAR are useful to determine the bounds at which the algorithms can create an impact, an important step would be to test the simulated drone policies on real hardware. Future work could explore utilizing the benefits gained from the training algorithm improvements to produce policies for real-world drones to follow. The decrease in training time in Incremental Learning and reduction of catastrophic forgetting impacts could reduce the time and effort needed to put RL software on physical hardware. On edge compute devices like drones, limited resources may make the storage-restricted approximations of IL-SOAR particularly effective, but additional experimental results are needed to verify this hypothesis.

Bibliography

[1] C. Barrado, R. Messeguer, J. Lopez, E. Pastor, E. Santamaria, and P. Royo, "Wildfire monitoring using a mixed air-ground mobile network," *IEEE Pervasive Computing*, vol. 9, pp. 24–32, Oct. 2010. Conference Name: IEEE Pervasive Computing.

[2] L. Merino, F. Caballero, J. R. Martinez-de Dios, J. Ferruz, and A. Ollero, "A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires," *J. Field Robotics*, vol. 23, pp. 165–184, Mar. 2006.

[3] K. D. Julian and M. J. Kochenderfer, "Distributed Wildfire Surveillance with Autonomous Aircraft Using Deep Reinforcement Learning," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1768–1778, 2019. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.G004106.

[4] R. N. Haksar and M. Schwager, "Distributed Deep Reinforcement Learning for Fighting Forest Fires with a Network of Aerial Robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1067–1074, Oct. 2018. ISSN: 2153-0866.

[5] N. Al-Thani, A. Albuainain, F. Alnaimi, and N. Zorba, "Drones for Sheep Livestock Monitoring," in *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, pp. 672–676, June 2020. ISSN: 2158-8481.

[6] H. Xiang and L. Tian, "Development of a low-cost agricultural remote sensing system based on an autonomous unmanned aerial vehicle (UAV)," *Biosystems Engineering*, vol. 108, pp. 174–190, Feb. 2011.

[7] T. Rakha, A. Liberty, A. Gorodetsky, B. Kakillioglu, and S. Velipasalar, "Heat Mapping Drones: An Autonomous Computer-Vision-Based Procedure for Building Envelope Inspection Using Unmanned Aerial Systems (UAS)," *Technology/Architecture + Design*, vol. 2, pp. 30–44, Jan. 2018. Publisher: Routledge _eprint: https://doi.org/10.1080/24751448.2018.1420963.

[8] T. Lee, S. Mckeever, and J. Courtney, "Flying Free: A Research Overview of Deep Learning in Drone Navigation Autonomy," *Drones*, vol. 5, p. 52, June 2021. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[9] A. Kumar, R. Krishnamurthi, A. Nayyar, A. K. Luhach, M. S. Khan, and A. Singh, "A novel Software-Defined Drone Network (SDDN)-based collision avoidance strategies for on-road traffic monitoring and management," *Vehicular Communications*, vol. 28, p. 100313, Apr. 2021.

[10] D. Hong, S. Lee, Y. H. Cho, D. Baek, J. Kim, and N. Chang, "Energy-Efficient Online Path Planning of Multiple Drones Using Reinforcement Learning," *IEEE Transactions on Vehicular Technology*, vol. 70, pp. 9725–9740, Oct. 2021. Conference Name: IEEE Transactions on Vehicular Technology.

[11] G. Zhan, X. Zhang, Z. Li, L. Xu, D. Zhou, and Z. Yang, "Multiple-UAV Reinforcement Learning Algorithm Based on Improved PPO in Ray Framework," *Drones*, vol. 6, p. 166, July 2022. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[12] A. Anwar and A. Raychowdhury, "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes using Transfer Learning," tech. rep., Oct. 2019. Publication Title: arXiv e-prints ADS Bibcode: 2019arXiv191005547A Type: article.

[13] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and Distributed Reinforcement Learning of Drones for Field Coverage," Tech. Rep. arXiv:1803.07250, arXiv, Sept. 2018. arXiv:1803.07250 [cs] type: article.

[14] G. Muñoz, C. Barrado, E. Çetin, and E. Salami, "Deep Reinforcement Learning for Drone Delivery," *Drones*, vol. 3, p. 72, Sept. 2019. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[15] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman, "MT-Opt: Continuous Multi-Task Robotic Reinforcement Learning at Scale," Tech. Rep. arXiv:2104.08212, arXiv, Apr. 2021. arXiv:2104.08212 [cs] type: article.

[16] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," Tech. Rep. arXiv:1802.01561, arXiv, June 2018. arXiv:1802.01561 [cs] type: article.

[17] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, (Montreal, Quebec, Canada), pp. 1–8, ACM Press, 2009.

[18] O. Buffet, A. Dutech, and F. Charpillet, "Incremental reinforcement learning for designing multi-agent systems," in *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, (New York, NY, USA), pp. 31–32, Association for Computing Machinery, May 2001.

[19] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient Surgery for Multi-Task Learning," p. 13, 2020.

[20] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," *Advances in neural information processing systems*, vol. 30, 2017.

[21] R. French, "Catastrophic Forgetting in Connectionist Networks," in *Trends in Cognitive Sciences - TRENDS COGN SCI*, vol. 3, Jan. 2006. Journal Abbreviation: Trends in Cognitive Sciences - TRENDS COGN SCI.

[22] C. Atkinson, B. McCane, L. Szymanski, and A. Robins, "Pseudo-rehearsal: Achieving deep reinforcement learning without catastrophic forgetting," *Neurocomputing*, vol. 428, pp. 291–307, Mar. 2021.

[23] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games," Tech. Rep. arXiv:2103.01955, arXiv, July 2022. arXiv:2103.01955 [cs] type: article.

[24] D. Weinshall, G. Cohen, and D. Amir, "Curriculum Learning by Transfer Learning: Theory and Experiments with Deep Networks," Tech. Rep. arXiv:1802.03796, arXiv, June 2018. arXiv:1802.03796 [cs] type: article.

[25] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to Fly – a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control," Tech. Rep. arXiv:2103.02142, arXiv, July 2021. arXiv:2103.02142 [cs] type: article.

[26] R. S. Allison, J. M. Johnston, G. Craig, and S. Jennings, "Airborne Optical and Thermal Remote Sensing for Wildfire Detection and Monitoring," *Sensors*, vol. 16, p. 1310, Aug. 2016. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

[27] M. L. Puterman, "Chapter 8 Markov decision processes," in *Handbooks in Operations*

Research and Management Science, vol. 2 of *Stochastic Models*, pp. 331–434, Elsevier, Jan. 1990.

[28] R. Bellman, "Dynamic Programming," *Science*, vol. 153, pp. 34–37, July 1966. Publisher: American Association for the Advancement of Science.

[29] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," p. 352, 2017.

[30] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems," Tech. Rep. arXiv:2005.01643, arXiv, Nov. 2020. arXiv:2005.01643 [cs, stat] type: article.

[31] Z. Ding and H. Dong, "Challenges of Reinforcement Learning," in *Deep Reinforcement Learning: Fundamentals, Research and Applications* (H. Dong, Z. Ding, and S. Zhang, eds.), pp. 249–272, Singapore: Springer, 2020.

[32] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016. Number: 7587 Publisher: Nature Publishing Group.

[33] Y. Li, "Deep Reinforcement Learning: An Overview," Tech. Rep. arXiv:1701.07274, arXiv, Nov. 2018. arXiv:1701.07274 [cs] type: article.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Tech. Rep. arXiv:1312.5602, arXiv, Dec. 2013. arXiv:1312.5602 [cs] type: article.

[35] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, Dec. 2020. Number: 7839 Publisher: Nature Publishing Group.

[36] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, Oct. 2017. Number: 7676 Publisher: Nature Publishing Group.

[37] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, pp. 350–354, Nov. 2019. Number: 7782 Publisher: Nature Publishing Group.

[38] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," Tech. Rep. arXiv:1912.06680, arXiv, Dec. 2019. arXiv:1912.06680 [cs, stat] type: article.

[39] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, May 1992.

[40] 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," Tech. Rep. arXiv:1602.01783, arXiv, June 2016. arXiv:1602.01783 [cs] type: article.

[41] 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," Tech. Rep. arXiv:1509.02971, arXiv, July 2019. arXiv:1509.02971 [cs, stat] type: article.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Tech. Rep. arXiv:1707.06347, arXiv, Aug. 2017. arXiv:1707.06347 [cs] type: article.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015.

[44] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," Tech. Rep. arXiv:1502.05477, arXiv, Apr. 2017. arXiv:1502.05477 [cs] type: article.

[45] L. Weng, "Policy Gradient Algorithms," Apr. 2018. Section: posts.

[46] W. Koch, R. Mancuso, R. West, and A. Bestavros, "Reinforcement Learning for UAV Attitude Control," Tech. Rep. arXiv:1804.04154, arXiv, Apr. 2018. arXiv:1804.04154 [cs] type: article.

[47] C. S. de Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. S. Torr, M. Sun, and S. Whiteson, "Is Independent Learning All You Need in the StarCraft Multi-Agent

Challenge?," Tech. Rep. arXiv:2011.09533, arXiv, Nov. 2020. arXiv:2011.09533 [cs] type: article.

[48] M. Minsky and S. Papert, "An introduction to computational geometry," *Cambridge tiass., HIT*, vol. 479, p. 480, 1969.

[49] K. Shmelkov, C. Schmid, and K. Alahari, "Incremental Learning of Object Detectors Without Catastrophic Forgetting," pp. 3400–3409, 2017.

[50] T. L. Hayes, K. Kafle, R. Shrestha, M. Acharya, and C. Kanan, "REMIND Your Neural Network to Prevent Catastrophic Forgetting," in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), Lecture Notes in Computer Science, (Cham), pp. 466–483, Springer International Publishing, 2020.

[51] J. Dong, Y. Cong, G. Sun, B. Ma, and L. Wang, "I3DOL: Incremental 3D Object Learning without Catastrophic Forgetting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 6066–6074, May 2021. Number: 7.

[52] D. Kim, J. Bae, Y. Jo, and J. Choi, "Incremental Learning with Maximum Entropy Regularization: Rethinking Forgetting and Intransigence," Tech. Rep. arXiv:1902.00829, arXiv, Feb. 2019. arXiv:1902.00829 [cs, stat] type: article.

[53] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence," pp. 532–547, 2018.

[54] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring Catastrophic Forgetting in Neural Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 2018. Number: 1.

[55] J. Ribeiro, F. S. Melo, and J. Dias, "Multi-task Learning and Catastrophic Forgetting in Continual Reinforcement Learning," Tech. Rep. arXiv:1909.10008, arXiv, Sept. 2019. arXiv:1909.10008 [cs, stat] type: article.

[56] O. Moulin, V. Francois-Lavet, P. Elbers, and M. Hoogendoorn, "Improving generalization to new environments and removing catastrophic forgetting in Reinforcement Learning by using an eco-system of agents," Tech. Rep. arXiv:2204.06550, arXiv, July 2022. arXiv:2204.06550 [cs] type: article.

[57] L. Panait and S. Luke, "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, Nov. 2005.

[58] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent Reinforcement Learning: An Overview," in *Innovations in Multi-Agent Systems and Applications - 1* (D. Srinivasan and L. C. Jain, eds.), Studies in Computational Intelligence, pp. 183–221, Berlin, Heidelberg: Springer, 2010.

[59] M. Samvelyan, T. Rashid, C. S. de Witt, G. Farquhar, N. Nardelli, T. G. J. Rudner, C.-M. Hung, P. H. S. Torr, J. Foerster, and S. Whiteson, "The StarCraft Multi-Agent Challenge," Tech. Rep. arXiv:1902.04043, arXiv, Dec. 2019. arXiv:1902.04043 [cs, stat] type: article.

[60] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen, A. C. Huang, Y. Wen, K. Hassanzadeh, D. Graves, D. Chen, Z. Zhu, N. Nguyen, M. Elsayed, K. Shao, S. Ahilan, B. Zhang, J. Wu, Z. Fu, K. Rezaee, P. Yadmellat, M. Rohani, N. P. Nieves, Y. Ni, S. Banijamali, A. C. Rivers, Z. Tian, D. Palenicek, H. b. Ammar, H. Zhang, W. Liu, J. Hao, and J. Wang, "SMARTS: Scalable Multi-Agent Reinforcement Learning Training School for Autonomous Driv-

ing," Tech. Rep. arXiv:2010.09776, arXiv, Oct. 2020. arXiv:2010.09776 [cs, eess] type: article.

[61] H. Ma, Y. Sun, J. Li, M. Tomizuka, and C. Choi, "Continual Multi-Agent Interaction Behavior Prediction With Conditional Generative Memory," *IEEE Robotics and Automation Letters*, vol. 6, pp. 8410–8417, Oct. 2021. Conference Name: IEEE Robotics and Automation Letters.

[62] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. G. Bellemare, and M. Bowling, "The hanabi challenge: A new frontier for ai research," *Artificial Intelligence*, vol. 280, p. 103216, 2020.

[63] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi, "QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning," Tech. Rep. arXiv:1905.05408, arXiv, May 2019. arXiv:1905.05408 [cs, stat] type: article.

[64] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," 2018.

[65] X. Wang, Z. Zhang, and W. Zhang, "Model-based Multi-agent Reinforcement Learning: Recent Progress and Prospects," Tech. Rep. arXiv:2203.10603, arXiv, Mar. 2022. arXiv:2203.10603 [cs] type: article.

[66] K. Zhang, Z. Yang, and T. Başar, "Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms," in *Handbook of Reinforcement Learning and Control* (K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, eds.), Studies in

Systems, Decision and Control, pp. 321–384, Cham: Springer International Publishing, 2021.

[67] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

[68] F. Christianos, G. Papoudakis, A. Rahman, and S. V. Albrecht, "Scaling Multi-Agent Reinforcement Learning with Selective Parameter Sharing," Tech. Rep. arXiv:2102.07475, arXiv, June 2021. arXiv:2102.07475 [cs] type: article.

[69] Z. Yang, K. Merrick, H. Abbass, and L. Jin, "Multi-Task Deep Reinforcement Learning for Continuous Action Control," pp. 3301–3307, 2017.

[70] M. S. Tomov, E. Schulz, and S. J. Gershman, "Multi-Task Reinforcement Learning in Humans," p. 16, 2019.

[71] N. Vithayathil Varghese and Q. H. Mahmoud, "A Survey of Multi-Task Deep Reinforcement Learning," *Electronics*, vol. 9, p. 1363, Sept. 2020. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

[72] S. Zeng, A. Anwar, T. Doan, A. Raychowdhury, and J. Romberg, "A Decentralized Policy Gradient Approach to Multi-task Reinforcement Learning," Tech. Rep. arXiv:2006.04338, arXiv, May 2021. arXiv:2006.04338 [cs, stat] type: article.

[73] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. v. Hasselt, "Multi-Task Deep Reinforcement Learning with PopArt," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3796–3803, July 2019. Number: 01.

[74] S. Ritter, J. Wang, Z. Kurth-Nelson, S. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick, "Been there, done that: Meta-learning with episodic recall," in *Proceed-*

ings of the 35th International Conference on Machine Learning (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4354–4363, PMLR, 10–15 Jul 2018.

[75] L. Pinto and A. Gupta, "Learning to Push by Grasping: Using multiple tasks for effective learning," Tech. Rep. arXiv:1609.09025, arXiv, Sept. 2016. arXiv:1609.09025 [cs] type: article.

[76] A. Wilson, "Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach," p. 8, 2007.

[77] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, "Learning by Playing - Solving Sparse Reward Tasks from Scratch," Tech. Rep. arXiv:1802.10567, arXiv, Feb. 2018. arXiv:1802.10567 [cs, stat] type: article.

[78] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-Task Reinforcement Learning with Soft Modularization," p. 11, 2020.

[79] S. Thrun and L. Pratt, *Learning to Learn*. Jan. 1998.

[80] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, Oct. 2010. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[81] Y. Bengio, "Deep Learning of Representations for Unsupervised and Transfer Learning," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pp. 17–36, JMLR Workshop and Conference Proceedings, June 2012. ISSN: 1938-7228.

[82] M. Taylor and P. Stone, "An Introduction to Inter-task Transfer for Reinforcement Learning," *AI Magazine*, vol. 32, Mar. 2011.

[83] A. Lazaric and M. Ghavamzadeh, "Bayesian Multi-Task Reinforcement Learning," *Proceedings of the 27th International Conference on Machine Learning (ICML)*, Aug. 2010.

[84] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning," Tech. Rep. arXiv:1511.06342, arXiv, Feb. 2016. arXiv:1511.06342 [cs] type: article.

[85] E. Kaufmann, L. Bauersfeld, and D. Scaramuzza, "A Benchmark Comparison of Learned Control Policies for Agile Quadrotor Flight," Tech. Rep. arXiv:2202.10796, arXiv, Feb. 2022. arXiv:2202.10796 [cs] type: article.

[86] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards Continual Reinforcement Learning: A Review and Perspectives," Tech. Rep. arXiv:2012.13490, arXiv, Dec. 2020. arXiv:2012.13490 [cs] type: article.

[87] F. M. Castro, M. J. Marin-Jimenez, N. Guil, C. Schmid, and K. Alahari, "End-to-End Incremental Learning," pp. 233–248, 2018.

[88] J. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "LEARNING TO REINFORCEMENT LEARN," p. 17, 2017.

[89] T. Schaul and J. Schmidhuber, "Metalearning," *Scholarpedia*, vol. 5, p. 4650, June 2010.

[90] L. Weng, "Meta-learning: Learning to learn fast," *lilianweng.github.io*, 2018.

[91] S. Hochreiter, A. Younger, and P. Conwell, "Learning To Learn Using Gradient Descent," pp. 87–94, Sept. 2001.

[92] J. Schmidhuber, J. Zhao, and M. Wiering, "Simple principles of metalearning," *Technical report IDSIA*, vol. 69, pp. 1–23, 1996.

[93] L. Kirsch and J. Schmidhuber, "Meta Learning Backpropagation And Improving It," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 14122–14134, Curran Associates, Inc., 2021.

[94] S. Ritter, J. Wang, Z. Kurth-Nelson, S. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick, "Been There, Done That: Meta-Learning with Episodic Recall," in *Proceedings of the 35th International Conference on Machine Learning*, pp. 4354–4363, PMLR, July 2018. ISSN: 2640-3498.

[95] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," 2016.

[96] M. Körber, J. Lange, S. Rediske, S. Steinmann, and R. Glück, "Comparing Popular Simulation Environments in the Scope of Robotics and Reinforcement Learning," Tech. Rep. arXiv:2103.04616, arXiv, Mar. 2021. arXiv:2103.04616 [cs] type: article.

[97] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, Oct. 2012. ISSN: 2153-0866.

[98] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," Tech. Rep. arXiv:1705.05065, arXiv, July 2017. arXiv:1705.05065 [cs] type: article.

[99] I. Yoon, M. A. Anwar, R. V. Joshi, T. Rakshit, and A. Raychowdhury, "Hierarchical memory system with stt-mram and sram to support transfer and real-time reinforcement learning in autonomous drones," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 485–497, 2019.

[100] G. Silano, E. Aucone, and L. Iannelli, "CrazyS: A Software-In-The-Loop Platform for the Crazyflie 2.0 Nano-Quadcopter," in *2018 26th Mediterranean Conference on Control and Automation (MED)*, pp. 1–6, June 2018. ISSN: 2473-3504.

[101] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A Flexible Quadrotor Simulator," Tech. Rep. arXiv:2009.00563, arXiv, May 2021. arXiv:2009.00563 [cs] type: article.

[102] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," p. 8, 2021.

[103] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for Distributed Reinforcement Learning," Tech. Rep. arXiv:1712.09381, arXiv, June 2018. arXiv:1712.09381 [cs] type: article.

[104] Bitcraze, "crazyflie_simulation," Oct. 2022. [Source code]. original-date: 2022-02-16T20:51:28Z.

[105] Bitcraze, "Crazyflie 2.1." [Website]. https://www.bitcraze.io/products/crazyflie-2-1/.

[106] S. Huang, R. F. J. Dossa, C. Ye, and J. Braga, "CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms," Tech. Rep. arXiv:2111.08819, arXiv, Nov. 2021. arXiv:2111.08819 [cs] type: article.

[107] Y. Tang, M. Valko, and R. Munos, "Taylor Expansion Policy Optimization," Tech. Rep. arXiv:2003.06259, arXiv, Mar. 2020. arXiv:2003.06259 [cs, stat] type: article.

[108] A. Gupta, A. Pacchiano, Y. Zhai, S. M. Kakade, and S. Levine, "Unpacking Reward Shaping: Understanding the Benefits of Reward Engineering on Sample Complexity," Tech. Rep. arXiv:2210.09579, arXiv, Oct. 2022. arXiv:2210.09579 [cs] type: article.

[109] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, pp. 3521–3526, Mar. 2017. Publisher: Proceedings of the National Academy of Sciences.