REINFORCEMENT LEARNING APPROACHES TO FLOCKING WITH FIXED-WING UAVS IN A STOCHASTIC ENVIRONMENT

APPROCHES D'APPRENTISSAGE PAR RENFORCEMENT POUR LES ESSAIMS DE DRONES À VOILURE FIXE DANS LES ENVIRONNEMENTS STOCHASTIQUES

A Thesis Submitted to the Division of Graduate Studies
of the Royal Military College of Canada
by

David Shao Ming Hung, B.A.Sc
Captain

In Partial Fulfillment of the Requirements for the Degree of
Master of Applied Science in Electrical Engineering

April, 2015

Acknowledgements

This work would not have been possible without the people that have helped and encouraged me along the way. I would like to especially thank Dr. Sidney Givigi, my supervisor, for his ongoing support, guidance, and for always pushing me to go further with my research. Dr. Francois Rivest for providing me with the necessary background to pursue my research in reinforcement learning. Dr. Steven Quintero for patiently explaining his research, and assisting me in building on top of his work. My better half, Sophie, for listening and commenting on each revision, and helping me gather my thoughts. And finally, my cat Guazi for keeping me company during the long nights while I procrastinated.

Abstract

In the past two decades, unmanned aerial vehicles (UAVs) have demonstrated their efficacy in supporting both military and civilian applications, where tasks can be dull, dirty, dangerous, or simply too costly with conventional methods. Many of the applications contain tasks that can be executed in parallel, thus can benefit in terms of effectiveness from deploying multi-UAVs working together as a force multiplier. However, to do so requires autonomous coordination among the UAVs, similar to swarming behaviors seen in animals and insects. This research looks at flocking with fixed-wing UAVs in the context of a model-free reinforcement learning problem, structured as a Markov decision process. The advantage of a model-free approach is that it can be applied to different platforms without the plant and disturbance models, which implies greater adaptability to changing environments and unforeseen situations. We propose two learning approaches that enable the agents, modeled as small fixed-wing UAVs, to learn control policies that facilitate flocking in a leader-follower topology, while operating in a non-stationary stochastic environment. The first approach is based on Peng's Q(λ) with a variable learning parameter, which learns through direct reinforcement learning. The second approach is based on Sutton's Dyna-Q where on-line learning, model learning, and planning are integrated to improve sample efficiency. Our approaches are compared to existing works by evaluating the respective policies at maintaining the desired flocking behavior according to a cost function. Simulation results demonstrate that with the two proposed learning approaches, the agents are able to learn policies that facilitate flocking with a single leader, more importantly, the agents are able to adapt their policies to non-stationary stochastic environments.

Résumé

Au cours des deux dernières décennies, les drones ont démontré leur efficacité au soutien d'applications tant militaires que civiles, où les tâches sont souvent ennuyeuses, dangereuses, ou tout simplement trop coûteuses avec des méthodes classiques. La plupart des applications contiennent des tâches qui peuvent être exécutées en parallèle. Ces tâches peuvent donc bénéficier d'un gain d'efficacité par le déploiement de plusieurs drones travaillant ensembles comme multiplicateur de force. Cependant, pour ce faire, il faut une coordination autonome parmi les drones, semblable à la formation en volée observée chez les animaux et les insectes. Cette recherche porte sur la formation en volée de drones à voilure fixe dans le contexte d'un problème d'apprentissage par renforcement sans modèle, structuré comme un processus de décision markovien. L'avantage d'une approche sans modèle est qu'elle peut être appliquée à différentes plates-formes sans modèle environnemental, ce qui implique une plus grande adaptabilité à l'évolution de l'environnement et aux situations imprévues. Nous proposons deux méthodes d'apprentissage qui permettent aux agents, modélisés comme des petits drones à voilure fixe, d'apprendre les politiques de contrôle qui facilitent la formation en volée dans une topologie meneur-suiveur, tout en fonctionnant dans un environnement stochastique non stationnaire. L'algorithme dans la première approche est basé sur la méthode d'apprentissage Q (λ) de Peng avec un paramètre d'apprentissage variable. Le second algorithme est basé sur la méthode Dyna-Q de Sutton où l'apprentissage en ligne, l'apprentissage du modèle, et la planification sont intégrés pour améliorer l'efficacité de l'utilisation des échantillons, ce qui accélère le processus d'apprentissage. Nos approches sont comparées à des œuvres existantes par l'évaluation des politiques respectives au maintien de la formation en volée selon une fonction de coût. Les résultats des simulations démontrent que pour les deux approches proposées, les agents ont appris des politiques qui facilitent la formation en volée. De plus, les agents ont pu adapter leurs politiques aux environnements stochastiques non stationnaires.

Contents

List of Tables

List of Figures

List of Symbols

S	a finite set of states
s	state
A	a finite set of actions
a	action
$P(s, a, s')$	state transition probability function
$R(s, a, s')$	reward function
r	reward
$\pi(s, a)$	policy function for state s and action a
t	time
T	terminal time
γ	discount rate
$V(s)$	value function
$Q(s, a)$	action value function or Q-value
α	learning parameter or rate
λ	trace decay parameter

Problem Formulation

k	discrete time steps
ξ	UAV state or state of an agent
x	planar x position
y	planar y position
ψ	heading
ϕ	roll
$\mathbb{S}$	n-sphere
$\mathbb{R}$	real numbers
z	system state
$\mathcal{Z}$	set of all system states
C	set of roll commands
r	roll command/action
X	set of discretized values for x and y
Ψ	set of discretized headings
$U(r)$	roll action space
$\eta_x, \eta_y, \eta_\psi$	disturbance terms in UAV kinematic model
$\mathbb{Z}$	integers
$P(z'\|z, r)$	state transition probability function
$\mathcal{N}$	normal distribution

$\hat{e}, \overline{e}, \check{e}$	roll error trajectory for $+15°$,$0°$,$-15°$
ρ	euclidean distance between leader and follower
β	tuning parameter for cost function
$g(z)$	cost function of being in state z
d	distance between follow to center of annulus
Λ	annulus
b_1, b_2	inner and outter radius of annulus
J	optimization criterion
E	expectation
$Q(z, r)$	Q-value for state z and roll action r
ς, p, ϱ	tuning parameters for variable learning parameter
$\xi_{m,j}$	UAV state or state of an agent in model learning
$x_{m,j}$	planar x position used in model learning
$y_{m,j}$	planar y position used in model learning
$\psi_{m,j}$	heading used in model learning
$\phi_{m,j}$	roll used in model learning
M_{in}	learned model of the environment
Ξ_j	sub-model in model learning
ξ_p	UAV state or state of an agent in the planner
x_p	planar x position used in the planner

y_p	planar y position used in the planner
ψ_p	heading used in the planner
ϕ_p	roll used in the planner
F	policy

Simulation Process & Results

F_{DP}	Quintero′s dynamic programming (QDP) policy
F_Q	Q-flocking policy in Exp. I
F_{Qns}	Q-flocking policy in Exp. II
F_{DQ}	Dyna-Q-flocking policy in Exp. I
F_{DQ50}	F_{DQ} with 50,000 learning agents in the planner
F_{DQ200}	F_{DQ} with 200,000 learning agents in the planner
F_{DQns}	Dyna-Q-flocking policy in Exp. II
F_{DQ50ns}	F_{DQns} with 50,000 learning agents in the planner
$F_{DQ200ns}$	F_{DQns} with 200,000 learning agents in the planner
Υ	a single trajectory out of the 1000 random trajectories used for evaluation
Γ	collection of Υ_1 to Υ_{1000}
Γ_{Ave}	mean of the collection of Υ_1 to Υ_{1000}
M_s	stochastic kinematic model used in Exp. I
M_{ns}	stochastic kinematic model used in Exp. II
μ	mean
σ	standard deviation

List of Acronyms and Abbreviations

DoF	degrees of freedom
DP	dynamic programming
Exp. I	Experiment I
Exp. II	Experiment II
MC	Monte Carlo
MDP	Markov decision process
MARL	multi-agent reinforcement learning
QDP	Quintero′s dynamic programming
RL	Reinforcement learning
TD	temporal-difference
UAV	unmanned aerial vehicle
ZOH	zero order hold

1 Introduction

Humans are curious beings. From the moment of birth, we are constantly exploring the environment around us, attempting to grasp the cause and effects of our actions. While we most often learn through supervised learning in schools and by our parents, we also have an innate nature to learn through association, demonstrations, and imitations. On the other hand, when faced with novel situations, we tend to learn through trial and error. These different ways of learning gives us insight on how to design intelligent machines that are capable of learning and adapting to their surroundings.

In robotics, we are faced with the challenge of developing accurate models of the environment so that we can design suitable controllers or policies. More often than not, these models are either unknown, non-linear, complex, or changing during operation. For this reason, we are interested in constructing machines that do not require a model to learn, and/or can construct their own models using information extracted from their interactions with the environment.

Reinforcement learning (RL) is a general class of machine learning that allows an agent to learn how to behave without a model of the environment. Learning is accomplished through interacting with the environment in the form of actions and rewards. Through trial-and-error, the agent learns the best action to take in order to maximize long-term rewards. This is different from other forms of learning. For example, in supervised learning the agent is presented with training data, such as input and output pairs that describe the correct outputs for the corresponding inputs. This is similar to having the answer key or an approximate solution to the problem. Another example is imitation learning, where the agent learns by observing a trainer that demonstrates the correct strategy. In both supervised and imitation learning, the agent is provided with relevant a priori knowledge or even a solution to the problem at hand. Rather than being instructed on what to do and told whether the act was right or wrong, RL methods rely on the rewards to assess its previous actions.

The aim of this research is to investigate the application of RL algorithms to flocking with fixed-wing unmanned aerial vehicles (UAVs) in a stochastic environment. Flocking is a swarming behavior exhibited by a group of birds foraging or in-flight, which can be emulated using rules for separation, alignment and cohesion [1]. This model has been widely adopted in the fields of computer generated animations [2, 3], mobile sensor networks [4, 5], control theory, and robotics [6–9]. Furthermore, the growth of UAV applications in both military and civilian domains [10, 11] has resulted in extensive research on the application of flocking to cooperative multi-UAV systems [12–16].

Specific to RL, flocking has been successfully demonstrated in simulation, where *particle-based agents* use Q-learning [17] to learn how to flock [18] and avoid predators [19]. More recently, La et al. [4,20] developed a hybrid system that combines a low level flocking controller with a high level RL module. The learning module determines safe spaces to navigate to, so that network topology and connectivity can be maintained while avoiding predators. Even though these works have incorporated RL methods in their approaches, there are two issues that have not been addressed. The first issue is learning in non-stationary stochastic environments, where the mean and variance of state transitions change over time. In these environments, the learned policies will change with the environment and do not necessarily converge to a single solution. The second issue is that the approaches thus far employ particle based agents, which are only applicable for omni-directional robots or rotary UAVs. In comparison to rotary aircrafts, fixed-wings have superiority over range, endurance, and payload capacity, but require a different approach to control in order to account for the non-holonomic constraints due to the dynamics of the aircrafts. So far, there has been no published results on the application of RL to flocking with fixed-wing UAVs in non-stationary stochastic environments.

In light of this, we intend to adopt the flocking framework proposed by Quintero et al. [14], and reformulate it in the context of a model-free RL task. Conveniently, the underlying Markov decision process (MDP) in [14] lends itself to RL, and the control policy generated using Quintero's approach provides a good benchmark, since the policies were field tested on small fixed-wing UAVs flocking and performing target tracking. In contrast to [14], which determines an optimal control policy *off-line* by assuming a priori knowledge of the system in the form of a model, we are relaxing the assumption of a priori knowledge of the model. Nevertheless, if the same model is used to simulate *on-line* learning, then we would expect the learned policies to be comparable to the policy generating using Quintero's approach.

1.1 Thesis Statement

Using Q-learning based algorithms, Peng's *Q-(λ)* [21] with a variable learning rate and Dyna-Q [22], agents modeled as small fixed-wing UAVs will learn a policy for flocking in a leader follower topology within a non-stationary stochastic environment.

1.2 Motivation

In the past two decades, UAVs have demonstrated their efficacy in supporting both military and civilian applications, where tasks can be dull, dirty, dangerous, or simply too costly with conventional methods. Examples include the use of UAVs for intelligence, surveillance, and reconnaissance [23], search and rescue, mining operations [24], and agriculture [25, 26]. Many of these applications contain tasks that are parallel in nature, thus can benefit from cooperation in terms of effectiveness [27]. Cooperative multi-UAV systems offer synergy [28], shorter task completion time (through parallelism) [27], greater spatial coverage, reduction in cost (through smaller, simpler, and cheaper robots), and increased robustness (through redundancy) [6]. One of the fundamental challenges of multi-UAV systems is to coordinate teams of UAVs to achieve a group objective or behavior [29], more importantly, to do this autonomously. Flocking, with its simplistic yet effective framework, has been widely adopted as the coordination scheme in multi-UAV systems.

Although there has been extensive research carried out on flocking in the fields of control theory and robotics [8], one of the underlying assumptions with traditional control systems is the availability of an accurate model of the plant [30]. Often, these models are either unknown, non-linear, complex, or changing during operation, thus hindering traditional analytical approaches [31]. As an alternative, machine learning techniques are being explored to provide robotic systems with tools to learn through interacting with the environment. In particular, RL allows the design of intelligent robots that can learn without models, thus making robots more versatile and adaptable to dynamic environments.

By taking an RL approach to flocking, we are investigating the feasibility of designing intelligent agents that can learn to flock. In addition, we are exploring the adaptive behavior of the agents, modeled as small fixed-wing UAVs, operating in non-stationary stochastic environments. Such methods could find use in coordinating teams of robots in dynamic environments, where the model may be unknown or incomplete.

1.3 Contributions

The contributions of this thesis are:

- a model-free RL approach to simulated flocking with small fixed-wing UAVs in a non-stationary stochastic environment by applying Peng's $Q(\lambda)$ [21] with a variable learning rate. The UAVs are assumed to be flocking in a leader-follower topology.
- a Dyna-Q approach to simulated flocking with small fixed-wing UAVs in a non-stationary stochastic environment, where the UAVs are assumed to be flocking in a leader-follower topology.
- a comparison of our approaches to Quintero's dynamic programming approach, herein known as the QDP approach. The comparison is based on the performances of the respective policies at maintaining the desired flocking behavior according to a cost function in the same environment.

1.4 Organization

This thesis is organized as follows. Chapter 2 provides a brief survey of reinforcement learning, including its theory and challenges. In addition, we look at various methods and applications of flocking, and end with providing some background on QDP. Subsequently, in Chapter 3, the flocking problem is defined in the context of RL, along with the notation to be used throughout the document. The chapter also introduces and discusses the proposed Q(λ) and Dyna-Q(λ) algorithms for solving the flocking problem. In Chapter 4, the simulation process, evaluation procedure, and simulation results are presented. Finally, conclusions and future work are presented in Chapter 5.

2 Background

In this chapter, we present an overview of RL, flocking, and the application of RL to flocking. We start in Section 2.1 with a brief introduction to RL including formal definitions of a Markov decision process, methods of solving RL problems, and challenges of RL. A short literature review on flocking is presented in Section 2.2, which leads up to Section 2.3, where the application of RL to flocking is discussed. In particular, we provide the necessary background on Quintero et al.'s work in [14] to differentiate our approach from theirs.

2.1 Reinforcement Learning

A critical requirement in developing control algorithms is a plant model for the system to be controlled. Often, these models are either unknown, nonlinear, and/or complex, thus hindering traditional analytical approaches [31]. As an alternative, machine learning techniques can provide robotic systems with the tools to learn control policies without a priori knowledge of the plant. In particular, RL is a general class of machine learning algorithms that aims at enabling an agent to learn how to behave in an incompletely known environment. Interactions with the environment involve the agent selecting actions that trigger a transition from one state to another, and observing the consequences in the form of rewards and punishments. The agent learns to associate the states, actions, and rewards through trial-and-error, and constructs a strategy that maximizes the expected long-term rewards. This is different from other forms of learning, such as supervised learning, where the agent is presented with training data, or imitation learning, where the agent learns by observing demonstrations of the correct strategy. In both supervised and imitation learning, the agent is provided with useful a priori knowledge. Rather than being instructed on what to do, and told whether the act was right or wrong, RL methods rely on the reinforcement signal to assess its previous

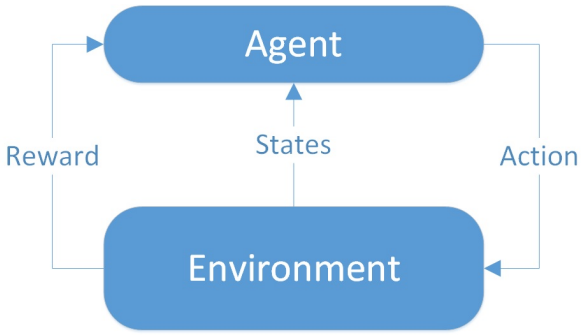

Figure 2.1: Interaction between the agent and the environment

actions.

2.1.1 Markov Decision Process

Classical RL problems assume a finite MDP that is commonly defined by a tuple $(S, A, P(s, a, s'), R(s, a, s'))$ [32], where

- S is a finite set of states,
- A is a finite set of actions,
- $P(s, a, s')$ is the state transition probability function or model that defines the probability of observing the next state $s' \in S$ after executing action $a \in A$ in state $s \in S$, and
- $R(s, a, s')$ is the reward function that specifies the reward after executing action $a \in A$ in state $s \in S$ and resulting in the next state $s' \in S$.

The RL interaction is illustrated in Figure 2.1. At each time step, the agent in state $s \in S$ chooses an action $a \in A$. As a consequence of that action, the agent moves to a new state $s' \in S$ with probability $P(s, a, s')$, and receives a reward $r \in R(s, a, s')$. Through iterations of this cycle, the agent learns a strategy, formally known as a policy $\pi(s, a)$, that specifies which action to take in each of the states, such that future rewards are maximized. For example, Figure 2.2 shows what a policy map looks like for a simple 2-D grid world, where the aim is to travel from the starting grid to the goal with the least amount of steps. The arrows indicate the corresponding actions that should be taken in each of the grids. To learn a policy, the agent would try different actions in each grid, and remember the rewards received with each action taken in each state. A common approach to decide which actions to take is to select the ones that result in the highest return (i.e., sum of rewards from start to finish).

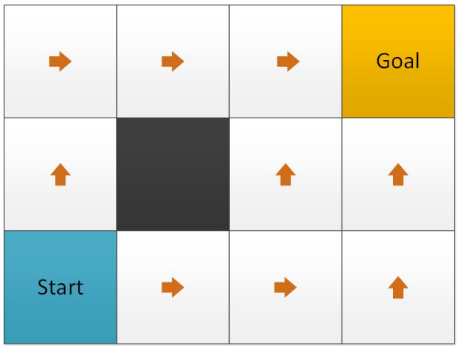

Figure 2.2: Policy map of a simple grid world

Returns are defined as the sum of the future rewards at time step t, and can be written mathematically as [32]

$$Return_t = r_{t+1} + r_{t+2} + ... + r_T = \sum_{k=0}^{T} r_{t+k+1} \quad , \tag{2.1}$$

where T is the terminal time, and $r_i \in R(s, a, s')$ represents the rewards received at time step i. The reward model in Equation 2.1 is used in applications that have a finite horizon such as plays of games [32]. In cases where the task is continuous (i.e., infinite horizon), future rewards are defined as [32]

$$Return_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad , \tag{2.2}$$

where the discount parameter γ is stipulated by $0 \leq \gamma \leq 1$. Discounting future rewards can be seen as a mathematical approach to bounding the infinite sum [33], or the economic notion of time value of money; a reward received now is worth more than one received later. A lower γ value implies a myopic agent that is only concerned with maximizing immediate rewards, which can potentially lead to poor performances if long-term rewards matter [34]. Conversely, as γ approaches 1, future rewards are weighted more strongly, thus the agent becomes farsighted.

In the context of a MDP, the states must contain all the information that an agent needs to make a decision. An approach to represent the *return* for each state is through value functions.

2.1.2 Value Functions

Value functions represent an estimation of how good it is for the agent to be in a certain state, or how good it is for the agent to take an action in a certain state. The measurement of how good a state or state-action is can be determined by the future rewards that are expected from that state, or from taking an action from that state, respectively. Formally, the state-value function for policy $\pi(s, a)$, assuming a discounted infinite horizon, is given by [35]

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left\{Return_t | s_t = s\right\} \\ &= E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right\} \quad , \end{aligned} \tag{2.3}$$

where t is any time step, and E_{π} denotes the expected value given that the agent follows policy π. Expressed in terms of the Bellman equation (Bellman 1957) Equation 2.3 becomes [35]

$$\begin{aligned} V^{\pi}(s) &= \sum_{a \in A} \pi(s, a) \sum_{s' \in S} P^a_{ss'} \left(R^a_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s\right\}\right) \\ &= \sum_{a \in A} \pi(s, a) \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V^{\pi}(s')\right) \quad , \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} \sum_{a \in A} \pi(s, a) = 1 \quad \forall s \in S; \quad and \\ \sum_{s \in S} P(s, a, s') = 1 \quad \forall s \in S, \quad \forall a \in A \quad . \end{aligned}$$

According to Equation 2.4, the expected value of the current state is defined in terms of the immediate reward determined by $R(s, a, s')$, and the discounted

future returns of possible next states $\gamma V^{\pi}(s')$, weighted by the respective transition probabilities $P(s, a, s')$, and the probability distribution over the agent's actions $\pi(s, a)$. If the agent was to follow an optimal policy π^*, Equation 2.4 becomes

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V^*(s') \right) \quad , \tag{2.5}$$

where the optimal state-value function $V^*(s)$ is now the Bellman optimality equation.

Likewise, the action-value function $Q^{\pi}(s, a)$ under policy $\pi(s, a)$ can be expressed as

$$Q^{\pi}(s, a) = E_{\pi} \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right\} \quad , \tag{2.6}$$

and its Bellman equation as

$$Q^{\pi}(s, a) = \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma Q^{\pi}(s', a') \right) \quad . \tag{2.7}$$

Lastly, the action-value function according to the Bellman's Principal of Optimality is defined as [32]

$$Q^*(s, a) = \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right) \quad , \tag{2.8}$$

where $Q^*(s, a)$ is the optimal action-value function.

The Bellman equation of $V^{\pi}(s)$ (as well as $Q^{\pi}(s, a)$) reveals the recursive relationship between the value of the current state $V^{\pi}(s)$ and the value of the successor states $V^{\pi}(s')$. This relationship divides the optimization problem into smaller sub-problems, and provides the means of solving it using algorithms categorized as dynamic programming (DP), Monte Carlo (MC) and temporal-difference (TD) learning [34].

2.1.3 Approaches to Solving RL Problems

The three main categories of algorithms used in solving RL problems are DP, MC and TD learning [31].

Dynamic Programming

DP algorithms are known as model based approaches, because they require a model of the transition probabilities $P(s, a, s')$ and the reward function $R(s, a, s')$ to calculate the value function. However, the models do not necessarily have to be provided beforehand; they may be learned from data [34]. The two common DP methods include *policy iteration* and *value iteration*.

Policy iteration comprises of two sub-processes, *policy evaluation* and *policy improvement*. The former evaluates policies by sweeping through and updating every state using [32]

$$V_{k+1}(s) = \sum_{a \in A} \pi(s, a) \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V_k(s') \right) \quad , \tag{2.9}$$

where k denotes the iteration number within policy evaluation. Each iteration of policy evaluation relies on the state-values estimated from the previous iteration. Consequently, evaluation ceases when $V_{k+1}(s) = V_k(s)$ (i.e., converge in the limit) or when $|V_{k+1}(s) - V_k(s)|$ is sufficiently small. The former condition states that the true state-values under the current policy have been found, while the latter condition means that the state-values are sufficiently close to their true values. Since convergence occurs in the limit, the second condition is used in practice as a stopping criterion [32].

Once the stopping criterion has been met, policy improvement greedily selects the best action in every state with respect to the latest state-values using [32]

$$\pi'(s) = \arg\max_a \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V(s') \right) \quad . \tag{2.10}$$

where $\pi'(s)$ is the new policy. Note that $\pi(s)$ implies deterministic actions, while $\pi(s, a)$ is a conditional probability, in which case the better actions would be assigned higher probabilities of being selected.

Policy iteration terminates when the new policy is equivalent to the previous policy, otherwise, the new policy is applied to a new round of policy evaluation.

In contrast to policy iteration, value iteration performs policy improvement after one iteration of policy evaluation. This allows the policy to quickly improve without going through several protracted iterations of policy evaluation in between policy improvements [32]. The update operation of value iteration is defined as

$$V_{k+1}(s) = \max_a \sum_{s' \in S} P(s, a, s') \left(R(s, a, s') + \gamma V_k(s') \right) \quad , \tag{2.11}$$

for all $s \in S$. Policy improvement is carried out by taking the maximum over all actions, so that once $|V_{k+1}(s) - V_k(s)|$ for all $s \in S$ are sufficiently small, a deterministic policy can be extracted based on Equation 2.10.

Monte Carlo

Similar to DP methods, MC methods compute the same value functions, and utilize concepts from policy evaluation and policy improvement. The main difference between the two is the way experiences or samples are gathered. Recall that DP methods are model-based, thus relies on the state transition model to generate successor states. MC methods on the other hand do not require the state transition model; the agent does not assume complete knowledge of the environment, and therefore MC methods are known as model-free approaches.

Policy evaluation is carried out by the agent interacting with the environment to gather sample sequences of states, actions, and rewards. Sampling occurs in episodes, where in each episode the agent starts in different initial states (i.e., exploring starts) and moves through the state space according to a given policy. As the agent moves through the state space, the visited states and actions carried out are stored in memory, along with the corresponding rewards received. At the end of the episode, for each state that was visited, the rewards collected subsequent to visiting the state are averaged and assigned to the state-action pair as its action-value $Q(s, a)$. Once the stopping criterion (e.g., $|Q_{k+1}(s, a) - Q_k(s, a)|$ for all $s \in S$ are sufficiently small) has been met, policy improvement can be used to greedily select the best action to take in every state with respect to the latest action-values, thus forming a new policy:

$$\pi'(s) = \arg\max_a Q(s, a) \tag{2.12}$$

for all $s \in S$.

Temporal Difference Learning

Learning by temporal differences TD [36] consists of ideas that are similar to DP and MC [32]. TD methods learn directly from experiences (i.e., model free), update state-values as soon as it is visited based in part on previous estimates of the state-value, and only "take into account the sampled successor

states rather than the complete distribution over successor states" [34]. The idea behind TD learning is that rather than waiting till the end of an episode to performs updates, an agent should learn on the go by updating the value of a state as soon as it is visited. By doing so, the agent is able to adjust its strategy incrementally to adapt to changes in the environment.

The most basic form of TD learning is TD(0), where the feedback (i.e., reward) used is from an one state transition of the underlying Markov Chain [37]. If the feedback is from multiple transitions similar to that of MC, then the algorithm is called TD(λ). The value function in TD(0) is updated using

$$V'(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \quad , \tag{2.13}$$

where $0 < \alpha \leq 1$ is the learning parameter or rate, $r \in R(s, a, s')$ is the reward received, $V(s)$ is the value of the current state, and $V(s')$ is the value of the next state. To put TD(0) into perspective, imagine an agent takes an action a that triggers a transition from s to s', and receives a reward $R(s, a, s')$. For the agent to learn from this experience, it has to first recall from its memory the values for the states s and s'. Then the agent can execute Equation 2.13 using the state-values and reward to determine a new estimate of the value of s. In layman terms, TD learning updates state-values (or action-values) by

$$NewEstimate = OldEstimate + StepSize[Target - OldEstimate] \quad ,$$

where $R(s, a, s') + \gamma V(s')$ is the *target*, $V(s)$ is the *old estimate* of s, and $V'(s)$ is the *new estimate* of s. $Target - OldEstimate$ is known as the TD error, and is weighted by a *step-size* known as the learning parameter or rate. This parameter describes the impact of the TD error on the current estimate. To ensure convergence of the state-values or action-values with probability one, the learning parameter must satisfy the following conditions [32]:

$$\sum_{k=1}^{\infty} \alpha_k(a) = \infty \qquad \sum_{k=1}^{\infty} \alpha_k^2(a) < \infty \tag{2.14}$$

The first condition ensures that the learning parameter is large enough to overcome initial conditions and random noise, and the second condition ensures that the parameter becomes small enough to assure convergence [32]. However, in non-stationary environments we want the agent to be constantly learning and adapting to the changes. This means that the estimates of the state-values should continue to change in response to the latest rewards. For that reason, the second condition is rarely used in applications where continuous learning is necessary, typically in response to non-stationary environments (e.g., robots moving in the real world).

In scenarios where the reward received are due to a series of transitions, the agent should remember which states have been visited and which actions were taken. For example, in the game of chess, a series of plays or states in the game may be required to set up for the capture of an opponent's piece. Upon receiving the reward for the capture, the agent may want to assign credit to the specific sets of states that made the capture possible. The general TD(λ) algorithm provides the means of assigning credit to previously visited states (similar to MC) through the use of eligibility traces. The update rule is defined as

$$V(s) = V(s) + \alpha(r + \gamma V(s') - V(s))e(s) \quad , \tag{2.15}$$

where $e(s)$ represents the eligibility trace of a given state. The lambda in TD(λ) represents the trace decay parameter used in eligibility traces, which is defined as

$$e(s) = \sum_{k=1}^{t} (\lambda\gamma)^{t-k} \delta_{s,s_k} \quad , \quad \text{where} \quad \delta_{s,s_k} = \begin{cases} 1 & \text{if } s = s_k \\ 0 & \text{otherwise} \end{cases} \quad , \tag{2.16}$$

where t is the current time step.

The eligibility of a state s determines how much of the reward should be assigned to that state, which according to Equation 2.16 is dependent on the recency and frequency of the state visit. If $\lambda = 0$, then eligibility traces are not used, and TD(λ) becomes the special case of TD(0). Although implementing eligibility traces creates additional complexity and computation, they offer several advantages such as faster learning, accounting for delayed rewards, and learning in non-Markovian environments [32].

For control applications, it is often more practical to learn the action-value function rather than the state-values functions. The latter requires a one-step look-ahead search to find the appropriate action to take, while an action-value function allows the agent to determine the greedy action by simply taking the argument of the maximum of $Q(s, a)$.

2.1.4 Q-Learning, Q(λ) and Dyna-Q

Q-learning (Watkins,1989) is one of the most important breakthroughs in RL, and widely used TD algorithms [31]. Rather than estimate the state-values, Q-learning estimates the action-values $Q(s, a)$, which are known as *Q-values*. Q-values are most commonly stored in a tabular format, where they can be updated using the algorithm:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a, s') + \gamma \max_a Q(s', a') - Q(s, a)] \quad . \tag{2.17}$$

By taking the maximum *Q-value* of the next state, the algorithm becomes exploration insensitive under the assumption that each state-action pair is visited an infinite number of times, and that the learning parameter α is decreased appropriately over time [38]. Exploration insensitive means that the agent can act sub-optimally in favor of exploration (i.e., off-policy), and still be in pursuit of learning the optimal action-values. This is particularly useful in non-stationary environments where exploration is constantly required to detect changes, but the optimal action values still have to be learned.

Unlike supervised learning, RL agents must explore their environment to acquire information about the rewards and the behavior of the system. This problem becomes a balancing act between playing it safe by taking known actions with known rewards, or being adventurous by taking actions randomly, which may lead to higher rewards or undesirable states such as falling off a cliff or crashing. There are several exploration methods available.

The simplest method is to take only the actions with the highest expected return. This is known as the greedy policy. The drawback to being greedy is that the agent becomes stuck in what it thinks is the optimal policy, where in fact it is only sub-optimal.

To avoid being trapped in a sub-optimal policy, the agent has to explore other options even if the other options have a lower expected return. This method is known as ϵ-greedy, where the agent takes a greedy action with a probability of $1 - \epsilon$, and a random action with probability ϵ. This strategy is simple and works well if exploration is required continuously, but it becomes inefficient once the optimal values have been found in a stationary environment; the agent would be exploring for no reason, and would not be maximizing its returns. One solution is to decrease ϵ as time goes on, but this will only work for learning in a stationary environment.

A more complex method to exploration is the softmax method, which selects an action probabilistically according to

$$Pr(a|s) = \frac{e^{Q(s,a)/T}}{\sum_{a_i \in A} e^{Q(s,a_i)/T}} \quad , \tag{2.18}$$

where T is a positive temperature parameter. Higher temperatures means all actions are nearly equi-probable, while lower temperatures means actions with higher values have greater chances of being selected. By tuning the temperature parameter at different stages of learning, we can control the amount of exploration, as well as directing it by selecting actions that seem more promising.

In handling delayed rewards, Watkins [17] and Peng [21] have each proposed methods of incorporating eligibility traces to Q-learning. Watkins's version of Q(λ) resets the traces if an exploratory or non-greedy action is taken. The reason for resetting the traces is that the experience from a non-greedy action cannot be used to evaluate a greedy policy. This makes sense in theory, however, if exploration occurs too often such as during early stages of learning or learning in a non-stationary environment, then much of the benefit of eligibility traces is lost [21]. For this reason, Peng proposed an alternative that makes no distinction between exploratory and greedy actions, but the updates are slightly more complex. The first update is the same as the one-step Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\delta_1 \quad , \tag{2.19}$$

where

$$\delta_1 \leftarrow R(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \quad , \tag{2.20}$$

The second update is for eligible state-action pairs, and is defined for all s, a as

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_2 Tr(s, a) \quad , \tag{2.21}$$

where

$$\delta_2 \leftarrow R(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1}) - \max_{a_t \in A} Q(s_t, a_t) \quad , \tag{2.22}$$

and

$$Tr(s, a) = \begin{cases} \gamma\lambda Tr(s, a) + 1 & \text{if } s = s_t, \text{ and } a = a_t; \\ \gamma\lambda Tr(s, a) & \text{otherwise.} \end{cases} \tag{2.23}$$

The time subscript t is used here to differentiate between the current state s_t, the next state s_{t+1}, and for all states s.

When greedy actions are taken, the temporal error in Equation 2.22 remains on-policy, meaning that the values used are according to the greedy policy being followed. On the other hand, if an exploratory (non-greedy) action is taken, the value of $R(s, a, s') + \gamma \max_{a' \in A} Q(s', a')$ becomes off-policy, while the value of $\max_{a \in A} Q(s, a)$ remains on-policy. The convergence of this

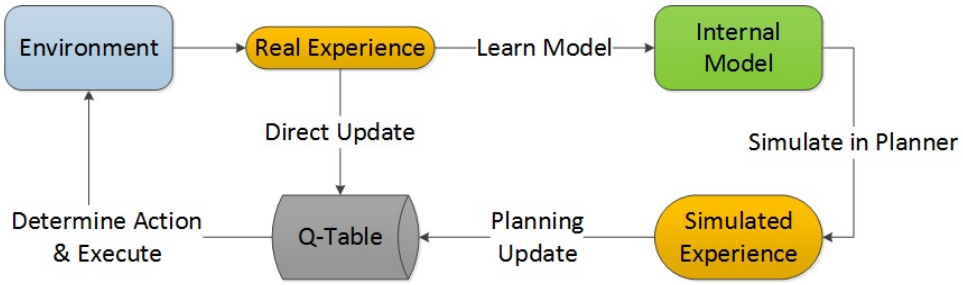

Figure 2.3: Dyna architecture

hybrid approach has yet to be proven, but studies have shown that Peng's Q(λ) performs better than Watkins's Q(λ) empirically [32].

The simplicity of Q-learning has made it one of the most popular RL algorithms. But to learn the optimal policy on-line, Q-learning requires a significant amount of experiences, especially if the state and/or action spaces are large. One way of improving sample efficiency is to learn a model that mimics the one-step input-output behavior of the environment using real experiences, and then to use this model to improve the policy through planning. Planning in this context refers to generating simulated experiences using the model, which can then be used to update Q-values. This approach of combining Q-learning, model learning, and planning is known as Dyna-Q [22]. As shown in Figure 2.3 both real and simulated experiences are used to update a Q-table.

2.1.5 Challenges of RL

Although the UAV agents in this thesis are simulated, the long term goal is to design algorithms that will enable them to learn and survive in the real world. To this end, we must understand the challenges of RL, and discover ways of overcoming them. Kormushev et al. (2013) [39] and Kober (2013) [34] have highlighted the main issues when solving real world problems in robotics. These include what Kober calls them the "Curse of Dimensionality" (Bellman, 1957), the curse of real-world samples, the curse of under-modeling and model uncertainty, and the curse of goal specification. Furthermore, specific to multi-robot systems, there are the design issues of specifying good multi-agent goals and coordination requirements [40]. In this section, we will touch on some of the main issues listed above, along with strategies to reduce the impact, or to solve them.

The "Curse of Dimensionality" was coined by Bellman (1957) [41] to describe the explosion of the state and action space in complex tasks. As a result, it becomes difficult or rather computationally expensive to sample or update every state; a requirement to ensure global optimality [34]. This is especially true in robotic systems where they operate in continuous states and actions, and possess several degrees of freedom. Strategies for managing large state spaces include hierarchical learning, task decomposition, and generalization. Hierarchical learning methods involve dividing the learning process into layers, and task decomposition is to shift complexity to a lower layer of functionality. Generalization uses methods such as function approximations and macro-actions that allow for compactness and transferable knowledge between states and actions that are similar [31, 33]. Furthermore, the use of policy-search has become a well established approach to avoid working in large state spaces. Policy-search involves searching directly in the policy-space (i.e., parameterization of all the possible policies), thus the dimensions are drastically reduced. However, the difficulty in policy-search lies in policy representation of the RL problem. See [39] for more details on policy-search and its challenges.

The curse of real-world samples refers to issues that arises when interacting with the physical world. The first issue is that environmental conditions are always changing, which causes robot dynamics to change. As a result, the learning process may not converge within accepted deviations [34]. In addition, environmental conditions in which a robot was learning in can rarely be reproduced exactly. This makes benchmarking algorithms rather difficult. The second issue of real-world samples is that robots must gather samples while maintaining certain levels of safety, and at the bare minimum be non-destructive [42]. The third issue is sensor data that are noisy, delayed and partial, a reality that algorithms must account for in gathering real-world samples. All of these issues depend on sample efficient algorithms that are able to learn from a small number of trials, thereby limiting expensive real-world samples [34]. Strategies for creating sample efficient algorithms typically require a model of the environment, whether it is through transfer learning or constructed with real-world samples. Transfer learning involves taking any form of knowledge and incomplete model about the task or even similar tasks, and using that to bootstrap the learning process. The concept of constructing a model using real-world samples while learning on-line was popularized by Sutton's Dyna architecture, discussed in the previous section (Dyna-Q).

The curse of under-modeling and model uncertainty addresses the shortfalls of robots that rely on transferred policies that were learned through simulations, which are used to offset the cost of real-world interaction. The

problem with simulations is that they are only as good as the underlying models. Small model errors will most likely lead to diverging results in the real world. For that reason, there are very few research that have successfully demonstrated real robots maintaining a high level of performance using simulated policies [34].

The curse of goal specification points out the difficulty in designing the reward function (i.e., reward shaping [43]) so that the desired behavior can be learned. In some scenarios, it may be sufficient to assign rewards upon task completion (e.g., winning a match, or finishing a maze), however, for tasks that have long completion times or are continuous for an indefinite amount of time, intermediate rewards should be used to assist in the learning process (e.g., penalties for taking extra moves to avoid longer paths).

In addition to the challenges discussed above, multi-agent reinforcement learning (MARL) naturally encounters design issues of incorporating multiple rewards and enabling cooperative learning. According to Busoniu et al. (2010) these algorithms are a fusion of temporal-difference learning (especially Q-learning), game theory, and more general direct policy search techniques [40]. Examples include optimal joint-action values, Team Q-learning (Littman, 2001) [44], and Distributed Q-learning (Lauer, 2000) [45]. In short, research in this area aims at combining experiences from multiple agents in an effective manner, with respect to policy convergence and optimality.

The inherent challenges of RL make the application of it to robotics rather difficult, and as a result, the "naive application of RL techniques to robotics is likely to be doomed to failure" [34]. Key principles that stem from years of RL research can be leveraged for success. These principles include effective representations, approximate models, and using prior knowledge [31, 34].

The notion behind effective representation is to represent the state space, action space and value functions efficiently. Examples of state and action space representation include splitting dimensions into regions, meta-actions (i.e., actions that are composed of a sequence of movements), and relational representations. Examples for value function representation include value-function approximation, generalizing to neighboring cells, and Gaussian process regression.

Simulating using approximate models, and bootstrapping with prior knowledge can dramatically help speed up and guide the learning process. More often then not, simulations of the task can be carried out using approximate transition dynamics. Subsequently, the simulated results can be used to jump-start the online learning process. Examples of where prior knowledge can come from include transferring policies from similar tasks, decomposing tasks into basic components to create building blocks, and demonstration followed by

imitation. In essence, use what we know to speed up the learning process.

These key principles embody countless years of RL research in discovering ingenious approaches to solving some of the major challenges of RL. However, applying RL techniques in the domain of robotics is not yet a straightforward undertaking, and many questions still remain open [34].

2.2 Flocking

In 1987, Reynolds [1] published an article on computer modeling of coordinate animal motion, in which he established the three basic rules to simulated flocking, now known as separation, alignment, and cohesion:

- Separation: steer to avoid collisions with nearby flockmates
- Alignment: steer towards the average heading of nearby flockmates
- Cohesion: steer to the center of nearby flockmates

Inspiration from Reynolds along with advancements in robotics have led to extensive research on the design, modeling, and analysis of flocking in the fields of robotics, control theory, and statistical physics [8].

In control theory, distributed control laws, supported with analytical proofs and simulated results demonstrate the feasibility of simulated flocking. Tanner et al. (2003) [46,47] proposed a stable control law that exhibits asymptotically stable flocking behavior in free space, based on the bearing, range, and velocity of neighboring robots. The proposed control law includes an attraction or repulsion term that is dependent on local distance, and an alignment term that is dependent on the local velocity. Olfati-Saber (2006) [16] presented several flocking algorithms, some of which consider flocking in free space with a virtual leader (i.e., objective), in the presence of obstacles, and with splitting/rejoining capabilities. The algorithms consist of a gradient-based attraction or repulsion term and a velocity matching term. La and Sheng (2011) [9] improved on Olfati-Saber's work by considering cluttered environments and noisy measurements, along with algorithms for target tracking. Moshtagh and Jadbabaie (2007) [48] proposed a control algorithm for non-holonomic vehicles that minimizes a misalignment potential, which results in velocity alignment. Provided that the underlying proximity graph is connected, a flocking behavior could be maintained.

Specific to ground robots, Mataric (1994) [49], Hayes and Dormiani-Tababaei (2002) [50], and Campo et al (2006) [51] have all experimented with wheeled robots, performing various forms of flocking using different hardware solutions for communication and networking in controlled indoor environments. Turgut

et al. (2008) [8] identified the main assumptions made in previous studies of self-flocking behaviors, which include that individuals can sense the range to the center of their neighbors, that there is at least one range reading per neighbor, and the need for designated leader or the use of a common goal. The authors removed these assumptions and successfully demonstrated their free-flocking algorithm on wheeled mobile robots, with complete on-board sensing.

In the domain of aerial vehicles, Welsby et al. (2001) flew three motorized blimp-like objects in an indoor environment to demonstrate flocking in three dimensions [52]. Hauert et al. (2011) deployed ten small fixed-wing UAVs outdoors to study the trade-off between communication range and flight dynamics. Their system performs like a flock, but separation for collision avoidance is maintained by each UAV flying at different altitudes [53]. More recently, Quintero et al. (2013) [14] experimented with flocking in a leader-follower topology, where the problem of determining the follower's policy is setup as a stochastic optimal control problem solved using DP. In addition to the novel algorithm, the policy generated was successfully tested on small fixed-wing UAVs flocking together to perform the task of target tracking. Vasarhelyi et al. (2014) implemented their flocking algorithm from [54], and created the first decentralized multi-copter flock that performed stable autonomous outdoor flight with up to ten agents [12].

In line with the intention of this research and the natural progression of flocking, it is a logical step to consider the design of *intelligent* agents by leveraging concepts of *machine learning*, thus enabling agents to learn and adapt to novel situations.

2.3 Application of RL to Flocking

The application of RL to flocking was first simulated by Tomimasu et al. (2005), where particle-based agents use Q-learning and potential forces to learn to flock [18]. Soon after, Morihiro et al. (2006) used the same approach to simulate particle-based agents learning to avoid predators while flocking [19]. More recently, La et al. (2013, 2015) published several works on the design of a hybrid system for cooperative flocking and learning to avoid predators [4, 20]. The system consists of a low level controller that facilitates flocking with particle-based agents and a high level RL module that learns how to avoid the predator while maintaining network connectivity. Cooperative learning is achieved through sharing Q-values, which expedites the learning process and results in a higher global reward.

A common theme in the application of RL to flocking is that the agents

are modeled as particles that maneuver by summing different velocity vectors that are based on distance and heading relative to other agents, as well as other vector forces generated to enable network connectivity and consensus. However, control solutions for particle based agents are only applicable to omni-directional robots and rotary based aircrafts. In comparison to rotary aircrafts, fixed-wings have superiority over range, endurance, and payload capacity, but requires a different approach to control in order to account for the non-holonomic constraints caused by the dynamics of the aircrafts. So far, there has been no published results on combining RL to flocking-with fixed-wing UAVs. Nonetheless, in [14], Quintero et al. proposed an approach to flocking with small fixed-wing UAVs, which could be reformulated in the context of RL.

2.3.1 Quintero's Approach to Flocking

In [14], Quintero et al. proposed a stochastic optimal flocking problem to which the optimal control policy facilitates flocking with fixed-wing UAVs in a leader-follower topology. The problem was setup as a MDP where the states were defined by the relative position of the follower to the leader, the actions were defined by discrete roll commands, and the state-transition function was defined by a stochastic discrete-time UAV kinematic model. The cost to the stochastic optimal control problem is a function of the distance and heading with respect to the leader. DP is used to minimize the cost criterion in a receding horizon fashion by performing value-iteration. The value function (i.e., value of each state) is updated on every execution of value-iteration. Each sweep through the state space is known as a backup, thus backing-up 100 times is equivalent to considering what the impact of the current action has on 100 times steps into the future. After a desired number of backups have been executed, the control policy is extracted from the value function by selecting and storing the action with the lowest cost for every state. The policy essentially specifies which roll action to take in a given state. In flight, the policy remains static and is used as a lookup table for action selection.

The issue with a static policy is that if the environment changes then the policy would become sub-optimal, or in the worst case, fail to provide proper control. For this reason, in this thesis we will reformulate Quintero's flocking problem into an RL problem, and use learning methods to create adaptive agents that can modify their policies as they interact with the environment in a simulated *on-line* setting.

For comparative purposes, we will also generate policies according to the QDP method. The policies generated using QDP provides a good benchmark, be-

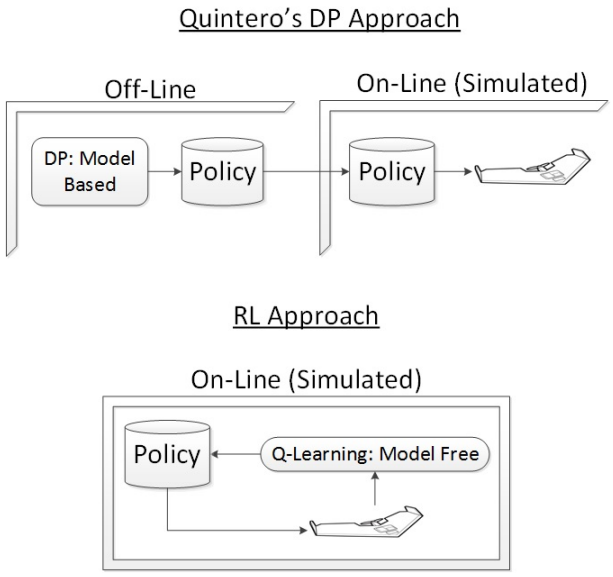

Figure 2.4: Difference between Quintero's DP approach and our RL approach

cause the approach calculates an optimal control policy, given that a stochastic model is available. If the same model is used to simulate *on-line* learning, then we would expect the resulting policy to be comparable to the policy generated using QDP. The difference between the QDP approach and our RL approach is illustrated in Figure 2.4. With RL we are simulating *on-line* learning, while with the QDP approach, we are performing *off-line* calculations to obtain a policy.

2.4 Summary

In this chapter, we presented an overview of RL including the definitions of a Markov decision process, methods of solving RL problems, and challenges of RL. Furthermore, we provided a brief review on flocking in terms of control and robotics. Finally, we looked at the application of RL to flocking, where we provided a short description of the QDP approach which forms the basis of our research. In the next chapter, we will formulate the stochastic optimal flocking problem in terms of an RL problem.

3 Problem Formulation

In this chapter, we formulate flocking as a model-free RL problem in the form of an MDP that is nearly identical to the one proposed in [14]. Furthermore, we define the two proposed learning approaches along with their respective algorithms. For consistency and ease of comparison, the notation used follows [14] and [32]. We start by introducing the flocking scenario followed by defining the components of the MDP, and finally, we present the two learning approaches.

3.1 Flocking Scenario

In our flocking scenario, the leader and the followers, also known as agents, are modeled as small fixed-wing UAVs flying at a constant average speed and fixed altitude. The leader has its own set of mission dependent control policy (e.g., target tracking, mapping, etc.), while the followers are to flock with the leader and minimize the overall cost, which is a function of distance and heading relative to the leader. To illustrate this idea, Figure 3.1 shows the leader centered on an annulus, and the followers positioned within the shaded region where the cost is low. If the followers move too far or too close to the leader, their respective costs will increase.

The agents maneuver by selecting a roll angle setpoint, which is regulated by an autopilot and is updated once every second (i.e., 1 second zero order hold (ZOH)). This implies that the simulation is run in discrete time-steps denoted by k. Collision avoidance between the agents is de-conflicted by operating at different altitudes, thus the same control policy can be used for each of the followers. Consequently, the objective is for the followers to learn a policy that flocks with the leader, while minimizing costs, and *without knowledge of the state transition model*. In solving this RL problem, the followers will learn a strategy that describes the best roll angle setpoint for a given state. When each follower adheres to the same strategy, the aggregate

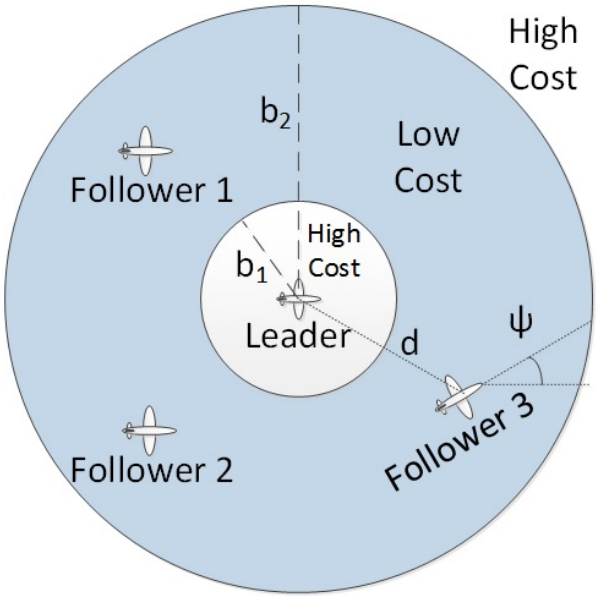

Figure 3.1: Top view of relationship between the leader & followers

behavior emulates flocking in a leader-follower topology.

3.2 Markov Decision Process

In the following subsections, we will present each component of the underlying Markov decision process to the RL problem.

3.2.1 State Representation

The individual agents are represented using a four state UAV model defined as $\xi := (x, y, \psi, \phi)$, where $(x, y) \in \mathbb{R}^2$ is the planar position, $\psi \in \mathbb{S}^1$ is the heading, and $\phi \in \mathbb{S}^1$ is the roll angle [14]; $\mathbb{S}^1$ (i.e., n-sphere where $n = 1$) represents the $(n + 1)$-dimensional Euclidean space, which in this case is a circle. The kinematic model used to simulate the transitions of the UAVs will be presented in section 3.2.3. For the purpose of flocking in a leader-follower topology, we are only concerned with the relative dynamics between the leader and the follower. Hence their respective UAV states ξ_l and ξ_f are combined to construct the system state space as $z := [z_1, z_2, z_3, z_4, z_5, z_6]^\top \in \mathcal{Z}$, where

$\mathcal{Z} := \mathbb{R}^2 \times [-\pi, \pi) \times \mathbb{S}^1 \times \mathbb{S}^1 \times C$ [14]; C is defined as

$$C := \{0^\circ, \pm 15^\circ, \pm 30^\circ\} \quad . \tag{3.1}$$

Similar to [14], the planar position of the follower relative to the leader represented by the pair (z_1, z_2) is defined as

$$\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} \cos\psi_l & \sin\psi_l \\ -\sin\psi_l & \cos\psi_l \end{bmatrix} \begin{bmatrix} x_f - x_l \\ y_f - y_l \end{bmatrix} \quad . \tag{3.2}$$

In addition, the remaining sub-states (z_3, z_4, z_5, z_6) shown in (3.3) represent the difference in the heading between the leader and follower, the follower's roll state, the leader's roll state, and the leader's roll command, respectively.

$$(z_3, z_4, z_5, z_6) := (\psi_f - \psi_l, \phi_f, \phi_l, r_l) \tag{3.3}$$

In the simulation, we assume that the leader's roll command is determined randomly, thereby introducing additional stochasticity to the problem. Although in practice, the leader's roll command would be mission dependent (e.g., target tracking, mapping). Moreover, we assume that the leader's state and roll command are broadcast through a wireless communication channel to the followers.

Lastly, the system state space is discretized as $\mathcal{Z} = X^2 \times \Psi \times C^3$, where $X = \{-150, -145, ..., 150\}$, and $\Psi = \{0^\circ, 15^\circ, ..., 345^\circ\}$. For clarity, X divides the planar surface into a 61 by 61 grid space, Ψ divides a circle that is centered on the leader into pie shaped sections each with a subtended angle of 15°, and C defines the discretized roll angle states.

3.2.2 Action Space

The fixed-wing agents maneuver by selecting their respective roll command $r \in C$, and holding them for one second or until the next command is selected. In order to mitigate any adverse effects on the mission caused by sharp changes in the roll, the next roll command is defined as $r' \in U(r)$ [14] where

$$U(r) := \{r, r \pm 15^\circ\} \cap C \quad . \tag{3.4}$$

The illustration on the top of Figure 3.2 depicts the five roll-angle states that the agents can be in, and the illustration on the bottom depicts the options for the next roll command; the agent can maintain current roll-angle or change by as much as ±15°. From a systems point of view, limiting the action space in our scenario helps alleviate the curse of dimensionality [41] at the cost of maneuverability.

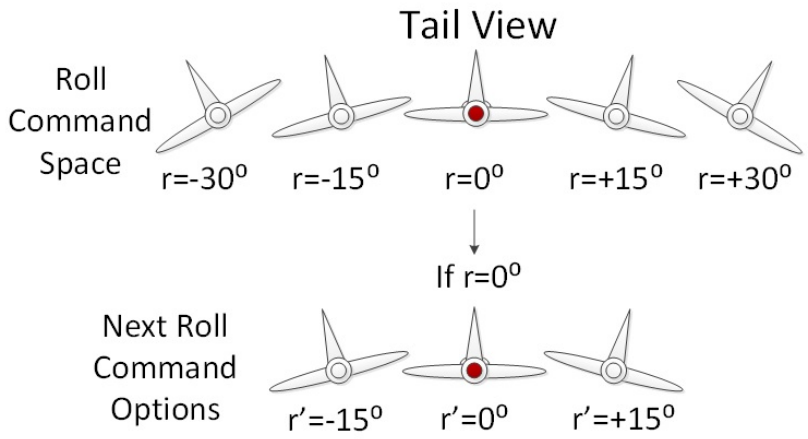

Figure 3.2: Illustration of the action space (Roll command)

3.2.3 State Transition Model

UAV kinematics are most accurately captured by a six degrees of freedom (DoF) aircraft model. However, by assuming our agents fly at a constant altitude and velocity, as well as with inertially coordinated turns, we can simplify the model down to four DoF. A coordinated turn is where the bank angle is set so that the centrifugal force acting on the aircraft is equal and opposite to the horizontal component of the lift acting in the radial direction [55]. This flight condition is commonly used in manned flights for passenger comfort. To make up for the loss of unmodelled dynamics and account for environmental disturbances, stochasticity is introduced in the roll, airspeed, and each of the sub-states of the model.

We adopt the continuous-time UAV kinematic model from [14], and include additional terms () in the first three states to represent disturbances that cause the xy-planar position and heading of the UAVs to change. By applying an one second ZOH to the roll command r, we create the discrete-time model, where the time steps are indexed by $k \in \mathbb{Z}_{\geq 0}$. This stochastic model generates state transitions for the individual agents, which are then combined using (3.2) and (3.3) to construct the system state transition model $P(z'|z, r_f)$. The

continuous-time model is defined as

$$\dot{\xi} = \frac{d}{dt}\begin{pmatrix} x \\ y \\ \psi \\ \phi \end{pmatrix} = \begin{pmatrix} s\cos\psi + \eta_x \\ s\sin\psi + \eta_y \\ -(\alpha_g/s)\tan\phi + \eta_\psi \\ f(\phi, r) \end{pmatrix} \quad , \tag{3.5}$$

where α_g is the acceleration due to gravity, and s is the nominal airspeed of the UAVs. The airspeed is drawn from a normal distribution $\mathcal{N}(\overline{s},\sigma^2)$, and held constant for the duration of the ZOH. In addition, the disturbance terms are drawn from normal distributions $\mathcal{N}(\overline{\eta}_x,\sigma_x^2)$, $\mathcal{N}(\overline{\eta}_y,\sigma_y^2)$, and $\mathcal{N}(\overline{\eta}_\psi,\sigma_\psi^2)$, respectively. Lastly, the function $f(\phi, r)$ defines the roll dynamics, which is sampled from a collection of stochastic second order roll trajectories.

To generate the trajectories, we simulate the initial condition response of the roll dynamics using a second-order system, where the undamped natural frequency ω_n and damping ratio ζ are selected based on the autopilot parameters of a small UAV [56]. For each roll command (increase by 15°, decrease by 15°, and maintain current roll), we collect the corresponding reference tracking error trajectories denoted as $\{\hat{e}_i(\tau)\}$, $\{\check{e}_i(\tau)\}$ and $\{\overline{e}_i(\tau)\}$, where $\tau \in [0, 1]$, $i \in \{1, ..., 1000\}$, and the accents indicate the respective roll commands that the error trajectories correspond to [14]. For example, a roll command to increase the roll angle by 15° would draw a sample from $\{\hat{e}_i(\tau)\}$. Figure 3.3 shows the collection of error trajectories for an increase of 15° in the roll angle setpoint. With the collection of error trajectories, the sample roll trajectories $\phi_i(\tau, r)$ are generated according to

$$\phi_i(\tau, r) = e_i(\tau) + r \quad . \tag{3.6}$$

Figure 3.4 depicts a sampled roll trajectory transitioning from 15° to 30°.

In total, there are five sources of randomness that can be modified in the stochastic kinematic model: η_x, η_y, η_ψ, the airspeed, and roll trajectory. With these sources of randomness, different models can be created to generate different set of successor states z'. Figure 3.5 illustrates the possible successor states for each roll command, where the agent is initially positioned at the origin ($x = 0, y = 0$) and facing the $+x$ direction. $M1$ represents a set of successor states generated with only stochasticity in the roll and airspeed, while $M2$ represents a set state transition model with stochasticity in the roll and airspeed, as well as non-zero means and variances for $\mathcal{N}(\overline{\eta}_x,\sigma_x^2)$, $\mathcal{N}(\overline{\eta}_y,\sigma_y^2)$, and $\mathcal{N}(\overline{\eta}_\psi,\sigma_\psi^2)$. Both models are stochastic in the sense that the exact successor states are non-deterministic, but it is the transition from one model to the

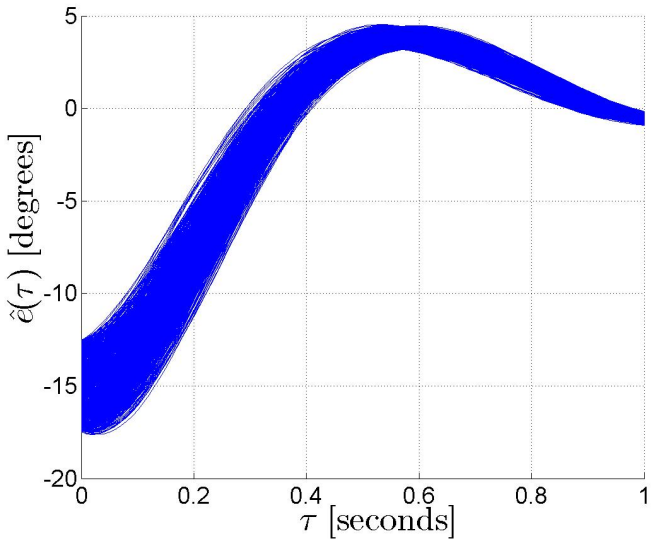

Figure 3.3: Simulated roll trajectories of +15° change in roll setpoint

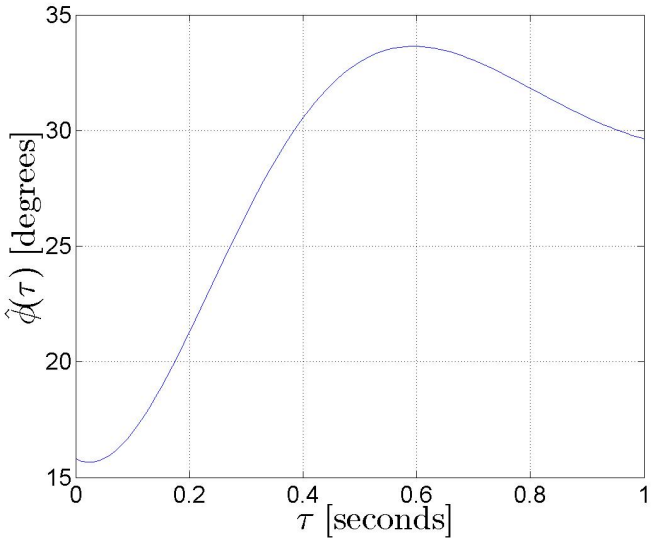

Figure 3.4: Sample roll trajectory changing roll-angle setpoint from 15° to 30°

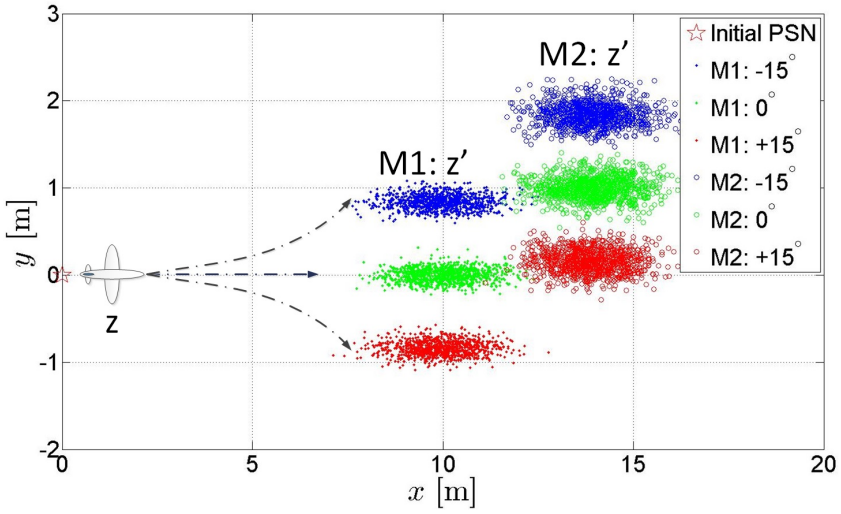

Figure 3.5: Collection of possible resulting UAV states z'. M1 represents the state transition model with only stochasticity in roll and airspeed. M2 represents the state transition model with stochasticity in roll and airspeed, as well as non-zero means and variances for $\mathcal{N}(\overline{\eta}_x,\sigma_x^2)$, $\mathcal{N}(\overline{\eta}_y,\sigma_y^2)$ and $\mathcal{N}(\overline{\eta}_\psi,\sigma_\psi^2)$.

other that creates the perception of the non-stationary environment that is referred to throughout this document.

3.2.4 Reward Scheme

To facilitate flocking, the reward function is represented as a cost function from [14], defined as

$$g(z) = \max\left\{d, \frac{b_1|z_3|}{\pi(1+\beta d)}\right\} \quad , \tag{3.7}$$

where

$$d = \max\left\{b_1 - \rho, 0, \rho - b_2\right\} \quad , \tag{3.8}$$

and β is a tuning parameter that modifies the impact of d. The parameters b_1 and b_2 define the inner and outer radius of an annulus Λ centered on the leader, shown in Figure 3.1, and given by

$$\Lambda = \{(z_1, z_2) \in \mathbb{R}^2 : b_1 \leq \rho \leq b_2\} \quad , \tag{3.9}$$

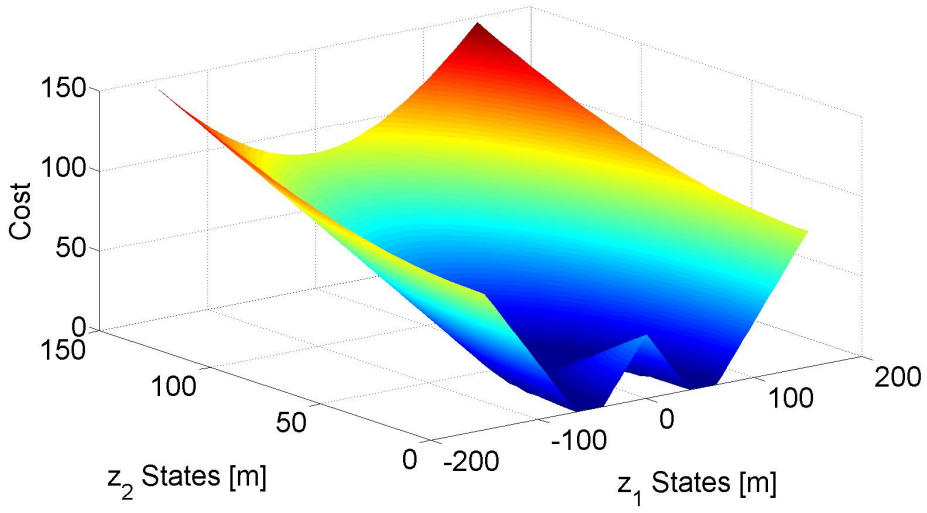

Figure 3.6: Cutout of the cost function for a subset of state-actions

where

$$\rho := \sqrt{z_1^2 + z_2^2} \quad . \tag{3.10}$$

The first argument in the maximization function of (3.7) governs the separation and cohesion rules of flocking, and the second argument assigns a cost to the absolute difference of the heading angles, thus reinforces the alignment rule. Figure 3.6 shows the cost plot relative to the leader who is centered at (0,0), As shown, the cost increases when the follower is near or far away from the leader. This is consistent with Figure 3.1 where the ring shaped region around the leader incurs the lowest cost.

3.3 RL Objective

With the elements of the MDP defined, it becomes clear that the RL problem is to learn the optimal control policy $F_k^* : \mathcal{Z} \rightarrow C, k \in \{0, ..., \infty\}$ that minimizes

$$J(z[0]) = E\left[\sum_{k=0}^{\infty} g(z[k])|z[0]\right] \ , \ \forall \ z[0] \in \mathcal{Z} \quad , \tag{3.11}$$

where $E[.]$ denotes expectation, and $z[k]$ refers to $z(t)$ at discrete time instances [14]. This minimization can be solved using DP to perform value iteration [14], provided that the model is a priori knowledge. In the context of RL, we assume that the learning agents do not have the model, and must employ a model-free approach, in this case Q-learning. The idea behind Q-learning is to incrementally estimate the value $Q^*(s,a)$ of a state-action pair (s,a), thereby assigning a value to the expected long-term reward that can be obtained by taking action a in state s. Taking into account of the reward being a cost, and replacing the standard notations with ours, $Q^*(z,r)$ can be determined by solving the Bellman [41] equation

$$Q^*(z,r) = \sum_{z'} P(z'|z,r) \left(g(z') + \gamma \min_{r'} Q^*(z',r') \right) \quad ,$$

where $P(z'|z,r)$ represents the state transition probability (i.e., from z to z' due to action r), and $0 \leq \gamma \leq 1$ is the discount factor. Subsequently, the optimal control policy can be determined by

$$F^*(z) = \arg\min_{r} Q^*(z,r) \quad .$$

In model-free RL problems, $P(z'|z,r)$ is unknown, and the only form of feedback is through rewards or punishments (e.g., cost). Hence, as the agent interacts with the environment $Q^*(z,r)$ is incrementally estimated using

$$Q'(z,r) \leftarrow Q(z,r) + \alpha[g(z) + \gamma \min_{r'} Q(z',r') - Q(z,r)] \quad ,$$

where $0 < \alpha \leq 1$ is the learning parameter. Ultimately, the goal is to find $Q^*(z,r)$, however, if $Q^*(z,r)$ is unknown in the sense that no closed form solution is available, then the next best solution is one that converges and provides a "good" policy, which may be sub-optimal. The alternative is to compare policies and to utilize the one the performs best, according to some benchmark.

3.4 Learning Approaches

The algorithms used in our learning approaches are based on Watkins' Q-learning with Peng's version of eligibility traces (i.e., Q(λ)) and Sutton's Dyna architecture [32]. The first approach, known as Q-flocking, applies Peng's Q(λ) with a variable learning rate to simulate learning through direct RL. The second approach, known as Dyna-Q-flocking, augments Q-flocking with

model learning and planning to simulate learning through indirect RL. In both approaches, the Q-values are stored in tabular format.

3.4.1 Q-Flocking

Q-flocking utilizes Peng's Q(λ) to update the Q-table. As discussed in Chapter 2, Q(λ) consists of two updates. The first update is an one-step update, defined using the new notation as

$$Q_{k+1}(z_k, r_k) = Q_t(z_k, r_k) + \alpha\delta \quad , \tag{3.12}$$

where

$$\delta = g(z_{k+1}) + \gamma \min_{r_{k+1}} Q(z_{k+1}, r_{k+1}) - Q(z_k, r_k) \quad . \tag{3.13}$$

The second update is based on the activity traces [21] (i.e., eligibility traces [32]) and is defined for all (z, r) pairs as

$$Q_{k+1}(z, r) = Q_k(z, r) + \alpha\delta' Tr(z, r) \quad , \tag{3.14}$$

where

$$\delta' = g(z_{k+1}) + \gamma \min_{r_{k+1}} Q(z_{k+1}, r_{k+1}) - \min_{r_k} Q(z_k, r_k) \quad , \tag{3.15}$$

and

$$Tr(z, r) = \begin{cases} \gamma\lambda Tr(z, r) + 1 & \text{if } z = z_k, \text{ and } r = r_k; \\ \gamma\lambda Tr(z, r) & \text{otherwise.} \end{cases} \tag{3.16}$$

The discrete time subscript k is used here to differentiate between the current state s_k, the next state s_{k+1}, and for all states s. The number of state-action pairs that need to be updated at each time step grows linearly with time, and in the worst case could encompass the entire state-action space [21]. In consideration of keeping the updating process at a manageable level, a five-step backup is used. This means that the previous five state-action pairs are remembered and assigned with the current TD error as shown in Equation 3.15.

According to Sutton and Barto (1998) [32], to ensure convergence with probability of one, the learning parameter must satisfy the following conditions:

$$\sum_{k=1}^{\infty} \alpha_k(r) = \infty \qquad \sum_{k=1}^{\infty} \alpha_k^2(r) < \infty \tag{3.17}$$

However, in a non-stationary environment it is desirable to have the estimated Q-values continue to vary in response to the most recent feedback, hence the second condition does not hold true [32]. When operating in non-stationary environments, it is common to use a fixed learning parameter, or adaptive strategies that determine the optimal size based on some meta-stepsize parameter [57]. The difficulty in tuning the learning parameter is that a high rate is needed for faster convergence, but it also increases the chances of divergence [57]. Without diving into the intricacies of determining an optimal learning parameter, we consider a rather simple method defined as

$$\alpha(\delta) = \max\left(\min\left(1, \left(\frac{|\delta|}{\varrho}\right)^{p}\right), \varsigma\right) \quad , \tag{3.18}$$

where $\varrho \in \mathbb{R}_{>0}$ and $p \in \mathbb{R}_{\geq 1}$ are tuning parameters, ς sets a lower bound, and δ is the temporal difference error (TD-error) defined by Equation 3.13. In essence, if there is a large difference between the feedback and what the learner already knows, then the difference (i.e., TD-error) is weighted more by using a larger α. Similarly, if the difference is small, then the estimated Q-values are most likely converging, or significant changes have yet been propagated (i.e., exploration issue), therefore the α should be relatively small. For practical purposes, the lower bound ς ensures that α does not become too small (i.e., learning becomes slow) and that $\alpha \neq 0$. The values for the parameters in Equation 3.18 are specified in Chapter 4.

Q-Flocking Algorithm

Figure 3.7 illustrates the interaction between the followers and the environment under the Q-flocking approach. The blocks highlighted in blue represent the real environment, such that given the current state of an agent and the its action as inputs, the environment would generate a state transition, and output the successor state. In addition, the blocks highlighted in orange represent the on-line learning processes occurring within the agent. Algorithm 1 shows the Q-flocking algorithm in episodic procedural form. For brevity, the following steps outline the follower's routine within each episode:

1. Receive the leader's state and combine it with own state to create a system state.
2. Select an action based on the ϵ-greedy method which is to explore or lookup the Q-table using the system state to find the greedy action.
3. Execute the action selected and observe the change in own state.
4. Receive the leader's state and combine it with own state to create a new system state.

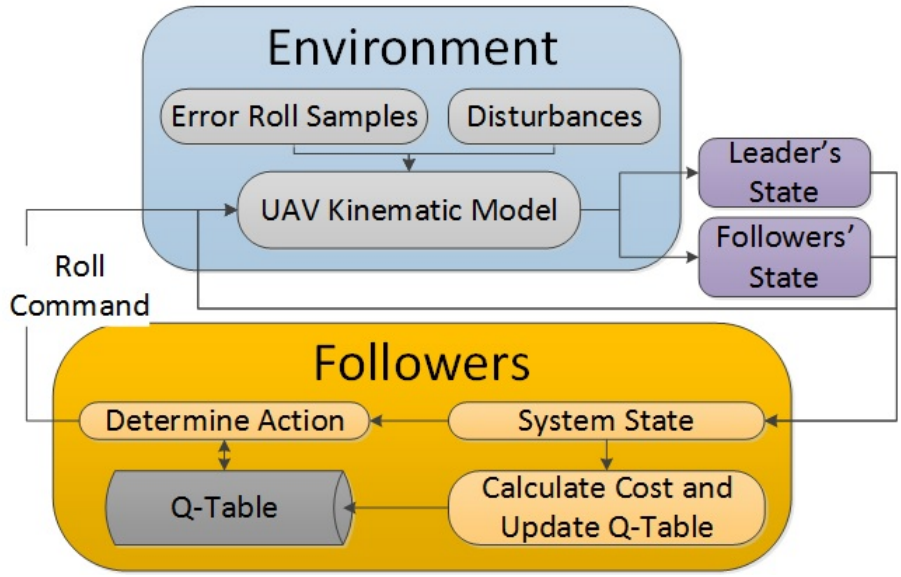

Figure 3.7: Agent-Environment interaction of Q-flocking

5. Calculate the cost using the new system state.
6. Update the Q-table table with the one-step update and the eligibility trace update.
7. Update the eligibility traces.
8. Go to step 2 and repeat until the end of the episode.

3.4.2 Dyna-Q-Flocking

Dyna-Q-flocking integrates model learning, planning, and Q-flocking, so that real experiences can be reused to expedite the learning process. The speed up is achieved by using the planner to generate simulate experiences to update the Q-table, in between real experiences. The advantage of model learning is that only a small number of on-line followers are required to gather state-transition experiences, and the advantage of using a planner is that the number of simulated learners can be scaled up with minimal cost (e.g., computation time).

Model Learning

In order to learn a model of the environment, we use the input and output states of the UAV kinematic model (i.e., environment) for both the leader and

Algorithm 1 Q-Flocking

initialize $Q(z, r)\ \forall\ z \in Z$, $r \in C$, γ, ϵ, T_{sim}, terminal conditions
  repeat(for each episode)
    ** Start On-Line Learning **
    initialize $z_0 \leftarrow (\xi_f, \xi_l)$ randomly
    while ($bound_{min} \leq \rho \leq bound_{max}\ ||\ k \leq T_{sim}$) **do**
      choose $r_{k,l} \in U(r_{k-1,l})$ randomly
      $r_{k,f} = \arg\min_r Q(z_k, r)$ or ϵ-greedy
      $(\xi_{k+1,f}, \xi_{k+1,l}) \leftarrow$ Sim Real UAV($\xi_{k,f}$,$r_{k,f}$,$\xi_{k,l}$,$r_{k,l}$)
      $z_{k+1} \leftarrow$ Create System State $(\xi_{k+1,f}, \xi_{k+1,l}, r_{k,l})$
      $\delta_k = g(z_{k+1}) + \gamma \min_{r_f} Q_k(z_{k+1}, r_f) - Q(z_k, r_{k,f})$
      $\delta'_k = g(z_{k+1}) + \gamma \min_{r_f} Q_k(z_{k+1}, r_f) - \min_{r_f} Q_k(z_k, r_f)$
      $\alpha \leftarrow$ Equation (3.18)
      for each state-action pair **do**
        $Tr(z, r) = \gamma\lambda Tr(z, r)$
        $Q_{k+1}(z, r) \leftarrow Q_k(z, r) + \alpha Tr(z, r)\delta'$
      end for
      $Q_{k+1}(z_k, r_{k,f}) \leftarrow Q_k(z_k, r_{k,f}) + \alpha\delta$
      $Tr(z_k, r_{k,f}) = Tr(z_k, r_{k,f}) + 1$
      $z_k \leftarrow z_{k+1}$; $k \leftarrow k + 1$; $\rho = \sqrt{z_1^2 + z_2^2}$
    end while
    ** End On-Line Learning **
  until desired number of episodes

followers to construct internal models, denoted as $M_{in,l}$ and $M_{in,f}$ respectively. As shown in the top left illustration of Figure 3.8, for each pair of input state $\xi := (x, y, \psi, \phi)$ and output state $\xi' := (x', y', \psi', \phi')$ from here on known as a sample, we calculate the planar translation, and the change in heading and roll, all relative to the input state using:

$$\begin{bmatrix} x_{m,j} \\ y_{m,j} \end{bmatrix} := \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} x' - x \\ y' - y \end{bmatrix}, \tag{3.19}$$

$$\psi_{m,j} := \psi' - \psi\ \ , \tag{3.20}$$

$$\phi_{m,j} := \phi'\ \ . \tag{3.21}$$

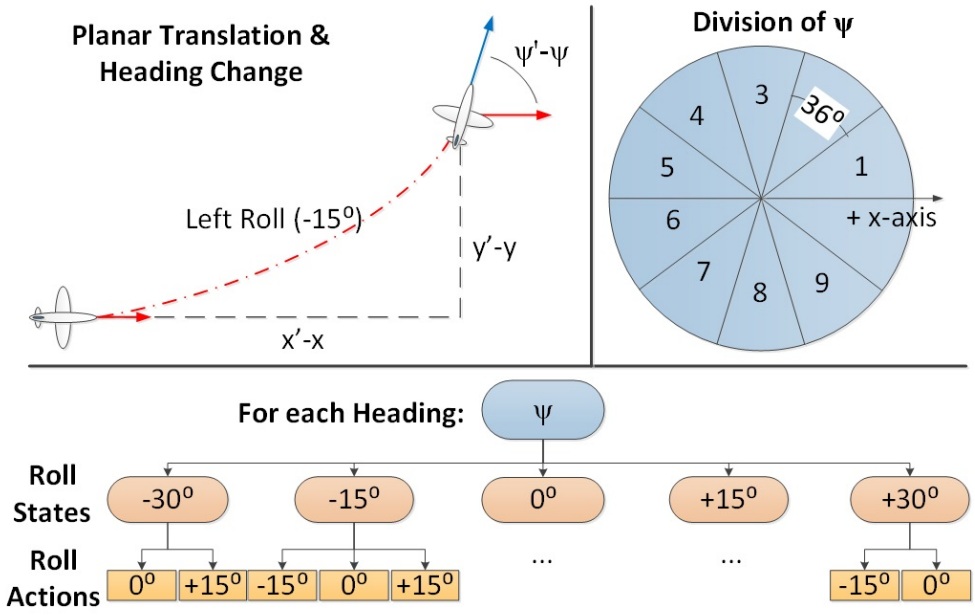

Figure 3.8: Components used for model learning

where m denotes internal model, and j indicates which sub-model the sample belongs to. For each sample, the resulting $\xi_{m,j} := (x_{m,j}, y_{m,j}, \psi_{m,j}, \phi_{m,j})$ is combined with previous samples from the same sub-model by using a cumulative weighted average. Each sub-model denoted as Ξ_j, is given by

$$\Xi_{j,k} := \frac{1}{2^{(k-1)}}(\xi_{m,j,1} + \sum_{h=2}^{k} 2^{(h-2)}\xi_{m,j,h}) \ , \tag{3.22}$$

where j denotes the sub-model number, k represents the current time step, and all of the sub-models are initialized with zeros at $k = 0$. The use of k to represent the current time step implies learning in a continuous manner such that k would keep incrementing until the end of the simulation. However, in a episodic setting, the model learned in one episode would simply carry over to the next episode, similar to how the Q-table would be carried over in between episodes of learning.

As previously defined in Subsection 3.2.2, there are five roll states ($C := \{0°, \pm 15°, \pm 30°\}$), and for each roll state, there are either two or three roll actions r' available to select, which is stipulated by $r' \in U(r)$ where

$$U(r) := \{r, r \pm 15^\circ\} \cap C \ .$$

The roll states and roll actions shown in the bottom illustration of Figure 3.8 depict 13 different transitions (e.g.,-15° to -30°) each requiring a sub-model to capture the transition. Furthermore, to account for directional disturbances (e.g., headwind or tailwind), the heading of the UAV has to be considered. For this reason, the heading $\psi_m \in \mathbb{S}^1$ is discretized into 10 pie-shaped sections, each with a subtended angle of 36° (see top right illustration of Figure 3.8). Hence in total, there are 130 sub-models (i.e. 13 transitions per discretized heading) to learn in order to construct an internal model M_{in} for the agents.

Planning

Planning involves simulating experiences using an internal model of the environment to update the Q-table. Conveniently, planning relies on the same processes as on-line learning, differing only in the source of their experience [32]. Generating state transitions using the internal model is accomplished by reversing the model learning process. Given a UAV state $\xi_p := (x_p, y_p, \psi_p, \phi_p)$ (p for planner), and a roll action r, the successor UAV state is defined as $\xi'_p := (x'_p, y'_p, \psi'_p, \phi'_p)$, where

$$\begin{bmatrix} x'_p \\ y'_p \end{bmatrix} := \begin{bmatrix} \cos\psi_p & -\sin\psi_p \\ \sin\psi_p & \cos\psi_p \end{bmatrix} \begin{bmatrix} x_{m,j} \\ y_{m,j} \end{bmatrix} + \begin{bmatrix} x_p \\ y_p \end{bmatrix}, \tag{3.23}$$

$$\psi'_p := \psi_{m,j} + \psi_p \ , \tag{3.24}$$

$$\phi'_p := \phi_{m,j} \ . \tag{3.25}$$

The roll action r, heading ψ_p, and roll state ϕ_p are used to lookup the corresponding sub-model Ξ_j containing $\xi_{m,j} := (x_{m,j}, y_{m,j}, \psi_{m,j}, \phi_{m,j})$. Once the leader and follower states have been calculated using the internal model, a system state can then be created, and the remaining processes are carried out identical to on-line learning.

Dyna-Q-Flocking Algorithm

Figure 3.9 illustrates the interaction between the followers and the environment under the Dyna-Q-flocking approach. The only differences to Q-flocking are the modeling and planning blocks, which are highlighted in green. The Dyna-Q-flocking algorithm is shown in episodic procedural form in Algorithm 2.

Algorithm 2 Dyna-Q-Flocking

initialize $Q(z, r)\ \forall\ z \in Z,\ r \in C,\ \gamma,\ \epsilon,\ T_{sim}$, terminal conditions

repeat(for each episode)

Start On-line Learning

initialize $z_0 \leftarrow (\xi_f,\ \xi_l)$ randomly

while $(bound_{min} \leq \rho \leq bound_{max}\ ||\ k \leq T_{sim})$ **do**

choose $r_{k,l} \in U(r_{k-1,l})$ randomly

$r_{k,f} = \arg\min_r Q(z_k, r)$ or ϵ-greedy

$(\xi_{k+1,f}, \xi_{k+1,l}) \leftarrow$ Sim Real UAV($\xi_{k,f}$,$r_{k,f}$,$\xi_{k,l}$,$r_{k,l}$)

$\Xi \leftarrow$ Update $M_{in}(\xi_{k+1,f}, \xi_{k+1,l}, \xi_{k,f}, r_{k,f}, \xi_{k,l}, r_{k,l})$

$z_{k+1} \leftarrow$ Create System State $(\xi_{k+1,f}, \xi_{k+1,l}, r_{k,l})$

$\delta_k = g(z_{k+1}) + \gamma \min_{r_f} Q_k(z_{k+1}, r_f) - Q(z_k, r_{k,f})$

$\delta'_k = g(z_{k+1}) + \gamma \min_{r_f} Q_k(z_{k+1}, r_f) - \min_{r_f} Q_k(z_k, r_f)$

$\alpha \leftarrow$ Equation (3.18)

for each state-action pair **do**

$Tr(z, r) = \gamma\lambda Tr(z, r)$

$Q_{k+1}(z, r) \leftarrow Q_k(z, r) + \alpha Tr(z, r)\delta'$

end for

$Q_{k+1}(z_k, r_{k,f}) \leftarrow Q_k(z_k, r_{k,f}) + \alpha\delta$

$Tr(z_k, r_{k,f}) = Tr(z_k, r_{k,f}) + 1$

$z_k \leftarrow z_{k+1}$; $k \leftarrow k + 1$; $\rho = \sqrt{z_1^2 + z_2^2}$

end while

End On-line Learning

Start Planner

initialize $z_{p,0} \leftarrow (\xi_{p,f},\ \xi_{p,l})$ randomly; reset k

while $(bound_{min} \leq \rho_p \leq bound_{max}\ ||\ k \leq T_{sim})$ **do**

choose $r_{k,l} \in U(r_{k-1,l})$ randomly

$r_{k,f} = \arg\min_r Q(z_k, r)$ or ϵ-greedy

$(\xi_{k+1,f}, \xi_{k+1,l}) \leftarrow$ Sim UAV with $M_{in}(\xi_{k,f}$,$r_{k,f}$,$\xi_{k,l}$,$r_{k,l})$

$z_{k+1} \leftarrow$ Create System State $(\xi_{k+1,f}, \xi_{k+1,l}, r_{k,l})$

... Same as On-line Learning

end while

End Planner

until desired number of episodes

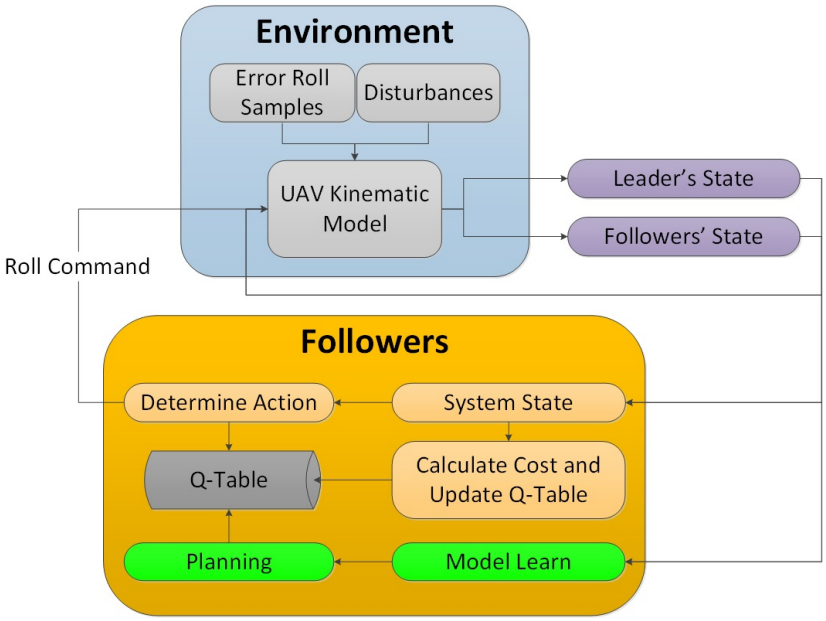

Figure 3.9: Agent-Environment interaction of Dyna-Q-Flocking

3.5 Summary

In this chapter, we formulated flocking as a model-free RL problem by defining the four components of the underlying MDP. The state space contains all the possible states to the problem, with each system state containing the information needed for a follower to select the next action. This information includes the relative position and heading of the followers with respect to the leader, the roll angles of the leader and follower, along with the leader's roll command. The action space is the roll angle commands that are available for the followers to select from. Each successive roll command can only be changed by $\pm 15°$, and is bounded by $\pm 30°$. The state-transition function is defined by a four DoF stochastic kinematic model of a small-fixed wing UAVs, and the reward function is defined as a cost function that considers the relative distance and heading to the leader. With the MDP defined, it is apparent that the RL objective is for the followers to learn a policy that minimizes the cost function, which in turn will facilitate flocking in a leader-follower topology. To enable the followers to learn, we proposed two learning approaches, Q-flocking and Dyna-Q-flocking. Q-flocking is based on Peng's Q(λ) augmented with a variable learning rate, while Dyna-Q-flocking combines Q-flocking with model

learning and planning to improve sample efficiency.

We simulated Q-flocking and Dyna-Q-flocking based on the RL problem formulated in this chapter. In the next chapter, we will present the simulation process and results, along with the evaluation procedure we used.

4 Simulation Process & Results

This chapter presents the process for simulating the RL problem defined in Chapter 3, the procedure for evaluating the learned policies, and the experimental results for Q-flocking and Dyna-Q-flocking.

4.1 Simulation Process

The aim of the simulation was to create an on-line episodic environment based on the RL problem defined in Chapter 3. In the simulation, multiple followers are simultaneously learning how to flock with a single leader, and are sharing the same Q-table. As shown by the flowchart in Figure 4.1, each episode begins with the initialization of random states and roll commands for all of the agents. Then the agents are simulated with the stochastic UAV kinematic model using the initial conditions. The kinematic model outputs successor states, which are used to create individual system states by combining each of the follower's state, the leader's state, and the leader's roll command. The system states are then used by the followers to calculate costs (Equation 3.7) when updating the Q-table, and to lookup the Q-table for the best action to take. The followers select their actions according to the $\epsilon - greedy$ method, and the leader selects its actions randomly. Once the actions have been determined, the process repeats itself until the end of the episode. The duration of each episode is limited to 30 time-steps, with each step representing one second wall-clock time. In addition, at each time-step a perimeter condition forces the followers to stop updating the Q-table when they are outside of X^2. This is to ensure that the value of the boundary states are not affected by followers wandering outside the perimeter.

The orange process and decision blocks shown in Figure 4.1 highlight the mechanisms within the learning agents (i.e., followers). In the same figure, the two green process blocks, model learning and planning, are specific to Dyna-Q-flocking. Model learning is shown as a sequential process within on-

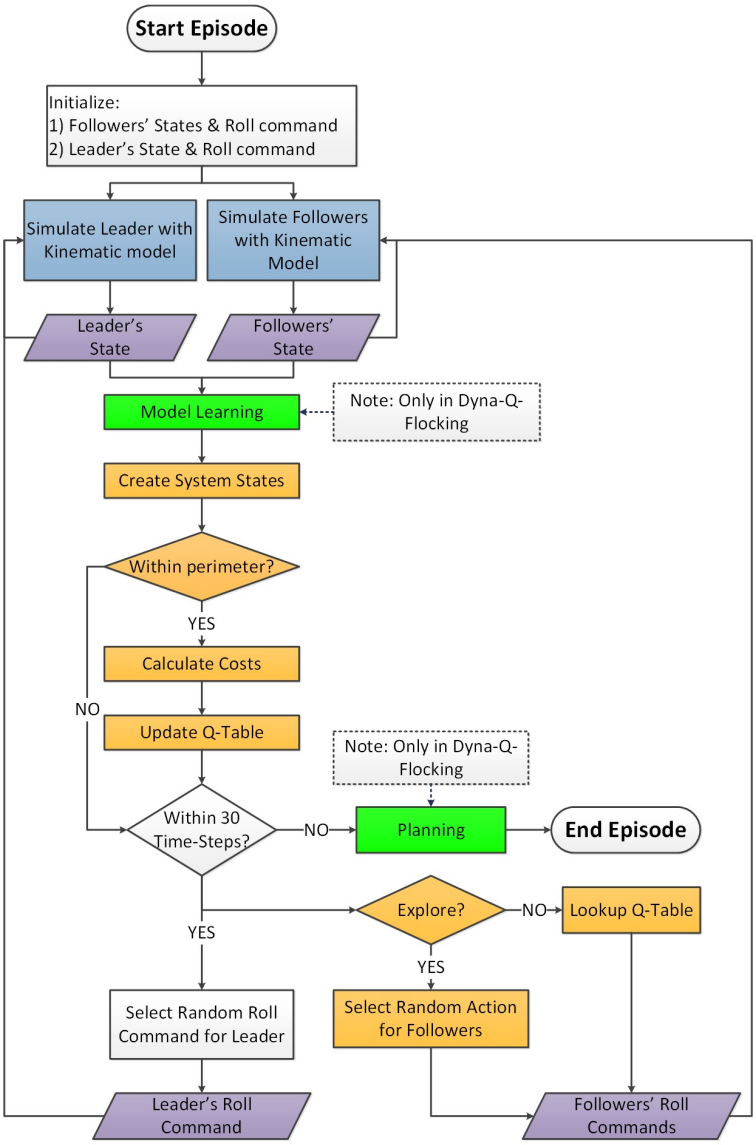

Figure 4.1: Flowchart of the simulation process

line learning, while planning occurs after on-line learning. In theory, model learning and planning can either run in the background or in parallel to on-line learning using multiple threads. Figure 4.2 provides a visualization of how the followers would physically flock with the leader.

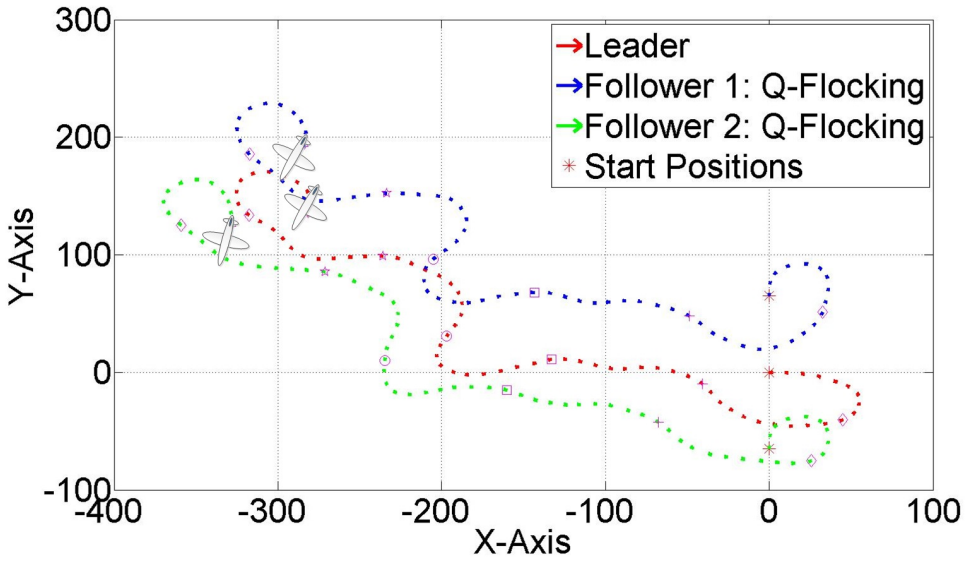

Figure 4.2: Simulated trajectories of two followers using a learned policy to flock with the leader.

4.2 Evaluation Procedure

Evaluation of the policies occur both during the learning process as the policies evolve, and at the very end when the policies converge. In both cases, the evaluation procedure is the same. To evaluate a policy, we simulate a follower using that policy to flock with a single leader over 1000 random trajectories to measure the costs incurred. The duration of each trajectory is 120 time-steps, and a single trajectory is denoted as Υ_n, with the subscript n indicating the trajectory number. The cost of each trajectory is averaged over the entire run (i.e., 120 time-steps), and at each time-step the cost is calculated using Equation 3.7. Figure 4.3 depicts a single run of a random trajectory for a leader and follower pair, where the leader's actions are random, and the follower looks up the best action to take according to a learned policy. The same set of random trajectories are used to assess all of the policies, and the collection of Υ_1 to Υ_{1000} for each policy is denoted as Γ, where $\Gamma(F)$ denotes the collection or set of average costs incurred by using policy F. The average

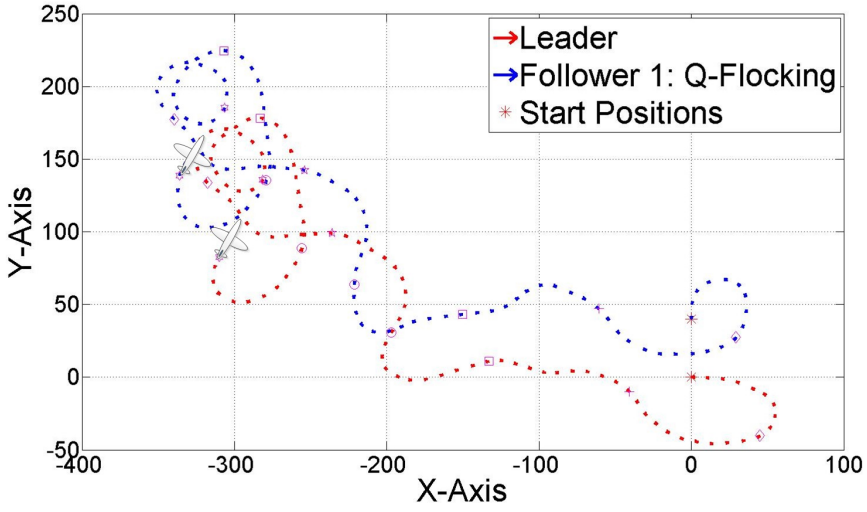

Figure 4.3: Plot of a single set of leader and follower trajectory. Markers are used to show relative positions of each UAV in time.

of Γ denoted as Γ_{Ave} is given as

$$\Gamma_{Ave} = \frac{1}{1000} \sum_{n=1}^{1000} \Upsilon_n \tag{4.1}$$

With the learning approaches, the policies are evaluated at the beginning of every episode to capture the evolution of the policies, which is reflected in the convergence of Γ_{Ave} over time. On the other hand, with the QDP approach every backup of value iteration generates a new policy, which is then evaluated. The learned policies using Q-flocking are denoted as $F_Q(\alpha)$, where α indicates the learning parameter used. Similarly, the learned policies using Dyna-Q-flocking are denoted as $F_{DQ}(\alpha)$. Policies generate using the QDP approach are denoted as $F_{DP}(\varpi)$, where $\varpi \in \mathbb{R}_{>0}$ indicates the number of backups.

To compare the policies, the mean and the standard deviation of Γ for each policy are computed after 1000 episodes, or in the case of QDP, the policy with the lowest Γ_{Ave} is used. In addition, t-tests are performed on Γ to assess the statistical significance of the average cost data. The t-test results indicate which policy performs better with a level of statistical confidence.

Figure 4.4 illustrates the overall policy evaluation and comparison process, where for example, six resulting policies are evaluated based on the same 1000

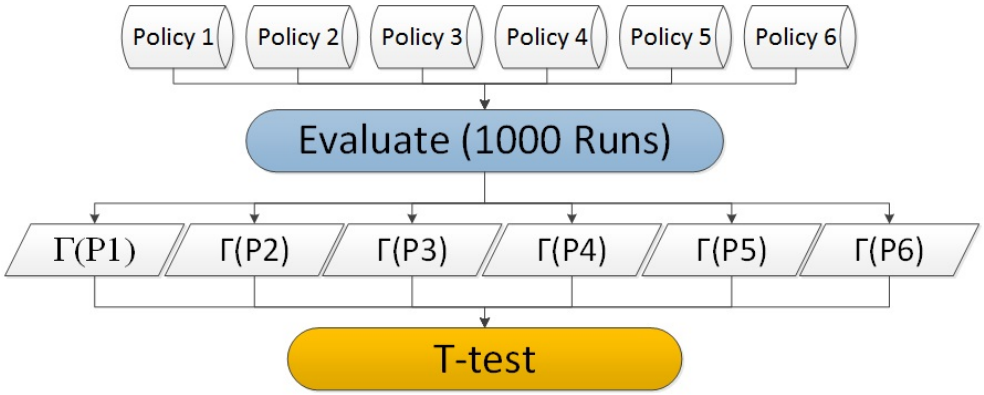

Figure 4.4: Policy evaluation and comparison

random trajectories. Subsequently, the resulting Γ data sets are compared against each other using paired t-tests.

4.3 Simulation Results

In this research we conducted two sets of experiments, where we simulated Q-flocking and Dyna-Q-flocking, and compared the resulting policies to ones generated using the QDP approach. The difference between the two experiments lies in the parameters used to define the stochastic UAV kinematic model. In Experiment I (Exp. I), the model denoted as M_1 accounts for stochasticity in the roll angle dynamics and the airspeed, while in Experiment II (Exp. II), additional disturbances (non-symmetrical) were introduced to the model denoted as M_2 by setting η_x, η_y, η_ψ and their respective standard deviations to non zero values. The followers in Exp. II are bootstrapped with the learned policies from Exp. I to simulate the perception of a non-stationary stochastic environment.

The first sub-table in Table 4.1 contains the general parameters that were used in the simulations, which include (from left to right) the discount rate, the decay rate, the inner and out radius of the annulus, the tuning parameter for the cost function, gravity, nominal airspeed, and the standard deviation of the airspeed. The second sub-table contains the disturbance parameters for the kinematic models used in each of the experiments, and the learning parameters that were experimented with. The third sub-table indicates the

Table 4.1: Simulation Parameters

General Parameters								
	γ	λ	b_1	b_2	β	α_g	$\overline{s}$	σ_s
Values	0.8	0.9	40	65	0.05	9.8	10	0.8

	Disturbance Parameters			Learning Parameters	
	$\overline{\eta}_x$, $\overline{\eta}_y$	$\overline{\eta}_\phi$	σ_x, σ_y, σ_ϕ	α: Static	α: (ϱ, p, ς)
Experiment I	0	0	0	$0.1, 0.8$	$(120, 3, 0.005)$
Experiment II	5	0.1π	0.01	$0.1, 0.8$	$(120, 3, 0.005)$

	Number of Learning Agents	
	On-Line (i)	Planner (i_p)
Q-flocking	50,000	N/A
Dyna-Q-flocking	5	50,000; 200,000

number of learning agents (i.e. followers) that were used in simulation for the different learning approaches. To expedite the learning process when using the Q-flocking approach, 50,000 followers were employed simultaneously to learn and update the massive Q-table (i.e., $61 \times 61 \times 24 \times 5 \times 5 \times 3 \times 3$). While in Dyna-Q-flocking, 50,000 followers were used in the planner and five were used in on-line learning (and model learning). Furthermore, to demonstrate the advantage of scaling with the planner, we also experimented with 200,000 followers in the planner, which was the maximum number of followers the planner could simulate without exceeding the computational time constraint of 25 seconds. This time constraint is a soft limit set on the planner to ensure that its runtime does not exceed 30 seconds, which is equivalent to 30 time-steps of on-line learning. By limiting the runtime of the planner, we are staying true to the possibility of running the planner in the background or in parallel to on-line learning.

4.3.1 Experiment I

The aim of Exp. I was to demonstrate the feasibility of using Q-flocking and Dyna-Q-flocking to learn how to flock in a stochastic environment defined by the model M_1. The experiment was divided into two parts. In part one, F_{DP} policies were generated according to the QDP approach and using the kinematic model M_1. This involved performing value-iteration (using DP) to propagate values through the state-action space. Each sweep through the

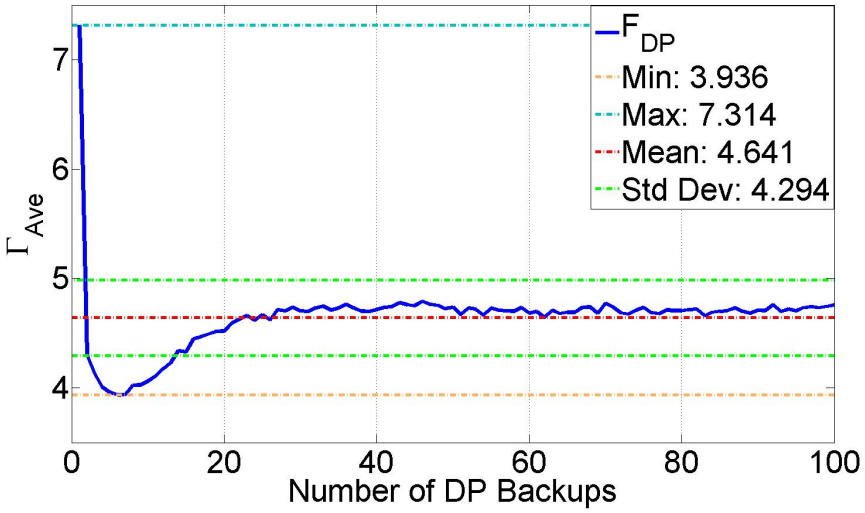

Figure 4.5: Plot of Γ_{Ave} for DP-value iteration backups using the QDP approach in Exp. I

state-action space is known as a backup, and a total of 100 policies were generated over the course of backing up 100 times.

According to Figure 4.5, the policy generated after the sixth backup $F_{DP}(6)$ incurred the lowest cost, and as the number of backups increased the average cost converged to approximately 4.7 (see Table 4.2). From the perspective of a model-based approach, the trend in Figure 4.5 suggests that the ideal planning horizon with the model M_1 is six backups, and that it becomes impractical to consider what the impact of the current action has beyond six time-steps into the future. Interestingly, the dip in the Γ_{Ave} shown in Figure 4.5 is counterintuitive in the sense that as the number of DP backups increase, the solution should gradually improve and converge. Upon closer examination of the QDP approach, it became clear that by taking the empirical average of the values of the successor states [14], the stochasticity between each successive run of value-iteration was trimmed. Consequently, this causes a biased representation of the expected values, as well as the state-transitions.

To help illustrate the effects of taking the empirical average of the values of the successor states, Figure 4.6 depicts the exponential growth of decisions to consider and the increase in stochasticity as the horizon increases (i.e., number of DP backups increases). Based on this figure, the transition from $K = 0$ to

$K = 1$ generated three collections of potential successor states shown in blue, green, and red. To properly capture the stochasticity in the stat-transitions as the horizon increase, the transition from $K = 1$ to $K = 2$ should generate three collections of successor states for every single point within the collections at $K = 1$. This approach though would be computationally expensive as the horizon increases. For example, if there are 1000 samples in each collection of successor state, then from $K = 0$ to $K = 1$ there would be 3000 samples (1000 for each action). Subsequently, from $K = 1$ to $K = 2$ there would be three million successor state samples (i.e., 3000×1000). This expansion of the samples should continue on with each successive backup, which makes the problem computationally expensive, and for this reason, averaging the samples is a more practical approach. However, by averaging the values of the successor states at each backup, the simulated (i.e., stochastic kinematic model) state-transition probability becomes a biased representation. As a result, some of the randomness in the overall problem are unaccounted. In spite of this, since $F_{DP}(6)$ incurred the lowest cost, its policy and Γ (i.e., 1000 average cost data set) were used as the benchmark in Exp. I.

Table 4.2: Γ_{Ave} of QDP policies in Exp. I

F_{DP}	μ	σ
1	7.314	3.750
2	4.301	2.415
3	4.129	2.269
4	4.006	2.205
5	3.960	2.124
6	3.936	2.106
7	3.938	2.143
8	4.024	2.296
9	4.025	2.235
10	4.062	2.283
⋮	⋮	⋮
100	4.756	2.574

In part two of Exp. I, we simulated Q-flocking and Dyna-Q-flocking with the parameters specified in Table 4.1. During the learning process, the policies were evaluated at every episode, and after 1000 episodes the learned policies against $F_{DP}(6)$. In the following sub-subsections, the results from simulating Q-flocking and Dyna-Q-flocking are presented and compared against $F_{DP}(6)$.

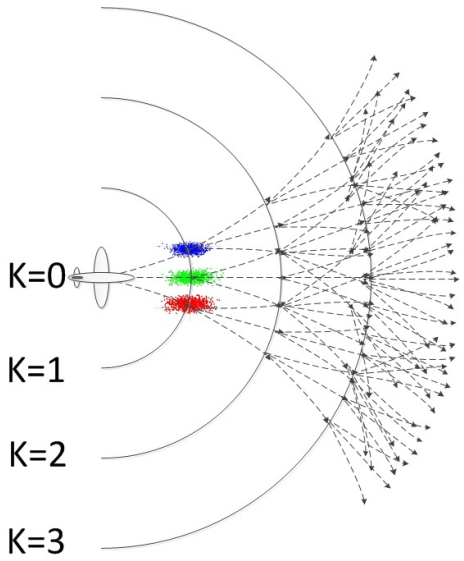

Figure 4.6: Exponential growth of decisions and transitions to consider as the horizon increases. Stochasticity is trimmed by taking the empirical average of the values of each collection of the successor states (shown in blue, green, and red).

Exp. I: Q-flocking

According to the learning curves shown in Figure 4.7, the followers were able to learn to flock with the leader using Q-flocking. Furthermore, the learning curves show that $F_Q(var)$ converged faster than $F_Q(0.1)$ during the initial transitional phase, moreover, $F_Q(var)$ converged to a lower Γ_{Ave} compared to $F_Q(0.8)$. This means that the proposed variable learning parameter offers faster convergence due to a larger learning rate during the transitional phases, as well as a lower convergence average cost due to a smaller learning rate as the TD-error decreases over time. The convergence values the learned policies are summarized in Table 4.3.

Figure 4.8 provides a visual comparison between two followers that utilize different policies to flock with the same leader; one follower used $F_Q(0.1)$ and the other used $F_{DP}(6)$. For comparison purposes, the followers started in the same state, and markers are used to show the relative positions of each UAV in 10 second increments. It is difficult to visually tell which follower incurs a lower cost, so to compare the polices, we needed to analyze the distribution of Γ for each policy.

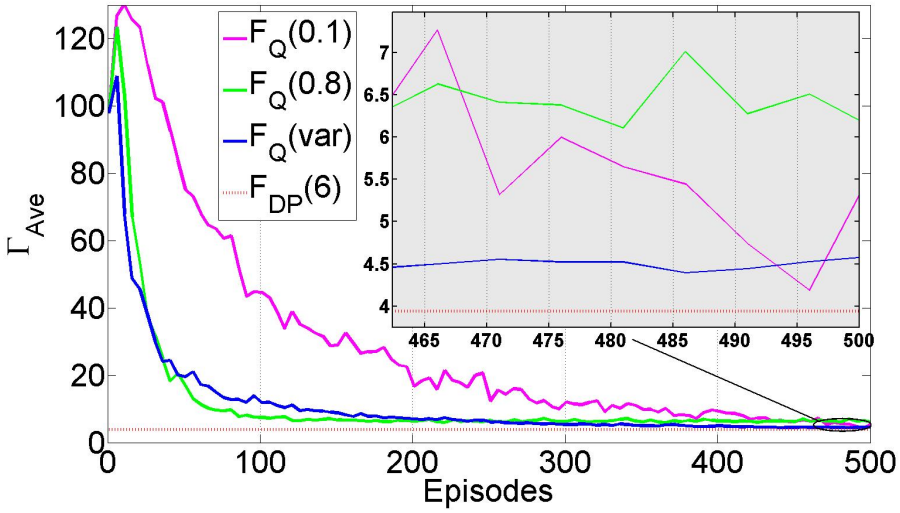

Figure 4.7: Learning curves for Q-flocking in Exp. I

According to the box plots shown in Figure 4.9, there are substantial overlapping between the distributions of Γ for each of the learned policies and $F_{DP}(6)$. For this reason, two-sample t-tests were performed on the data sets in order to statistically compare the performance of the policies. The following two sets of hypotheses denoted as H1 and H2, were considered for each of the learned policies:

- $H1_0$: The average cost incurred using F_Q and $F_{DP}(6)$ are the same.
- $H1_A$: The average cost incurred using F_Q and $F_{DP}(6)$ are not the same.
- $H2_0$: The average cost incurred using F_Q is larger than $F_{DP}(6)$.
- $H2_A$: The average cost incurred using F_Q is smaller than $F_{DP}(6)$.

The subscript 0 represents the null hypothesis, while the subscript A represents the alternative.

The t-test results for Q-flocking are compiled in Table 4.4. The first column lists the learned policies as a function of their respective learning parameters. In the second column, the t-values pertaining to each policy are shown. The t-values indicate the difference between the means of the two samples such that the larger the t-value, the larger the difference between the means. The third column shows the degrees of freedom, which relates to the number of values in the data sets; more data means higher degrees of freedom, as well as a smaller

Table 4.3: Γ_{Ave} of policies learned using Q-flocking in Exp. I after 1000 episodes

Method	μ	σ
$F_Q(0.1)$	3.768	2.274
$F_Q(0.8)$	6.436	2.957
$F_Q(var)$	3.809	2.224
$F_{DP}(6)$	3.936	2.106

Note: For F_Q, the values were sampled after 1000 episodes. For F_{DP}, the values were sampled after the number of backups as indicated.

sampling error. Both the t-value and degrees of freedom are used to determine the significance of the test, commonly known as the p-value. In practice, p-values smaller than 0.001 (i.e., the significance level) suggests *very strong evidence* against the null hypothesis H_0, and values smaller than 0.01 suggest *strong evidence*. For completeness, the paired difference results including the mean, the unpooled estimated standard deviation, and the confidence intervals (i.e., $100 \times (1 - 0.001)\%$) are also included in Table 4.4.

Table 4.4: T-test results for Q-flocking in Exp. I

$H1_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_Q(0.1)$	-1.712	1998	0.087	-0.168	2.192	-0.491	0.155
$F_Q(0.8)$	21.780	1998	1.529E-94	2.501	2.567	2.122	2.879
$F_Q(var)$	-1.309	1998	0.191	-0.127	2.165	-0.446	0.192

$H2_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_Q(0.1)$	-1.712	1998	0.044	-0.168	2.192	-Inf	-0.006
$F_Q(0.8)$	21.780	1998	1.000	2.501	2.567	-Inf	2.690
$F_Q(var)$	-1.309	1998	0.095	-0.127	2.165	-Inf	0.033

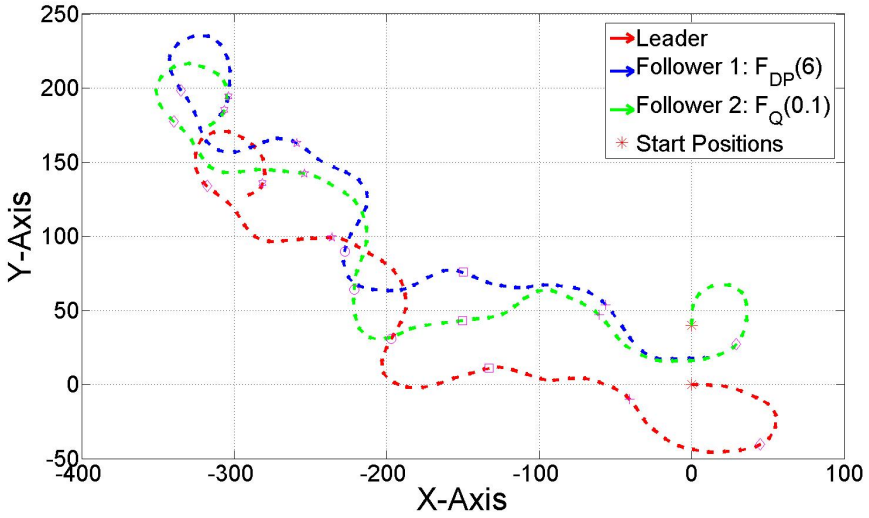

Figure 4.8: Trajectory plot of two followers flocking with a single leader in the environment defined by M_1. One follower is using $F_Q(0.1)$ (after convergence), and the other is using $F_{DP}(6)$. For comparison purposes, the followers both started in the same state, and markers are used to show the relative positions of each UAV in 10 second increments.

According to the results of the $H1$ test shown in Table 4.4, the p-values for $F_Q(0.1)$ and $F_Q(var)$ are both higher than 0.001, while the p-value for $F_Q(0.8)$ is less than 0.001. This means that there are no significant differences between the distribution of $\Gamma(F_{DP}(6))$, and the distribution of both $\Gamma(F_Q(0.1))$ and $\Gamma(F_Q(var))$. In other words, $F_Q(0.1)$ and $F_Q(var)$ are both comparable to $F_{DP}(6)$ in performance. Based on the results of the $H2$ test, we can confirm that on average $F_Q(0.8)$ incurred higher costs in comparison to $F_{DP}(6)$. Figure 4.10 illustrates the empirical cumulative distribution function for $\Gamma(F_Q(0.8))$, $\Gamma(F_Q(var))$, and $\Gamma(F_{DP}(6))$. From this graph, we can see that the average costs incurred by $F_Q(0.8)$ was higher than the average costs incurred by $F_{DP}(6)$, while the average costs incurred by $F_Q(var)$ was lower than $F_{DP}(6)$ in most parts of the graph, and higher near the tails.

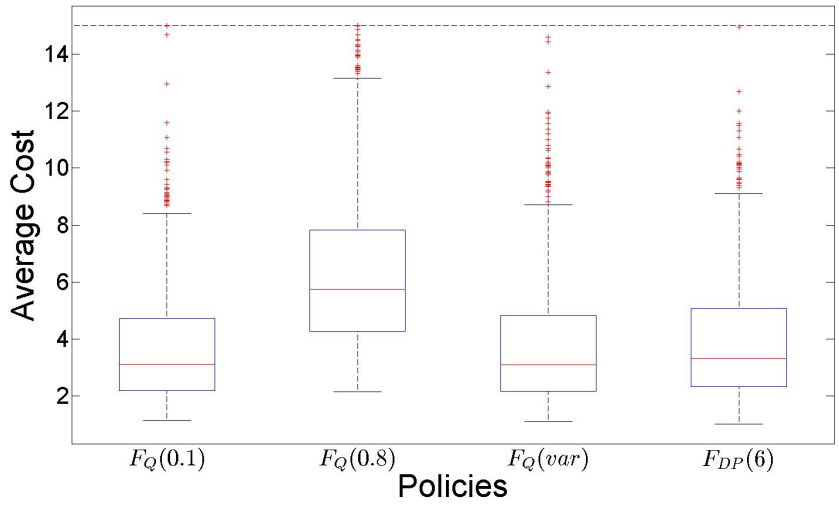

Figure 4.9: Box plot of Γ for all Q-flocking policies in Exp. I

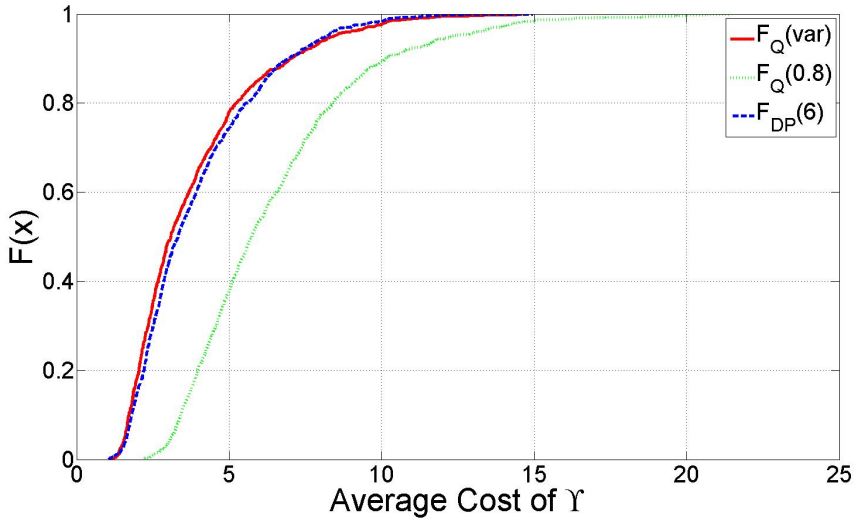

Figure 4.10: Empirical cumulative distribution function for Γ of $F_Q(var)$, $F_Q(0.8)$ and $F_{DP}(6)$ in Exp. I

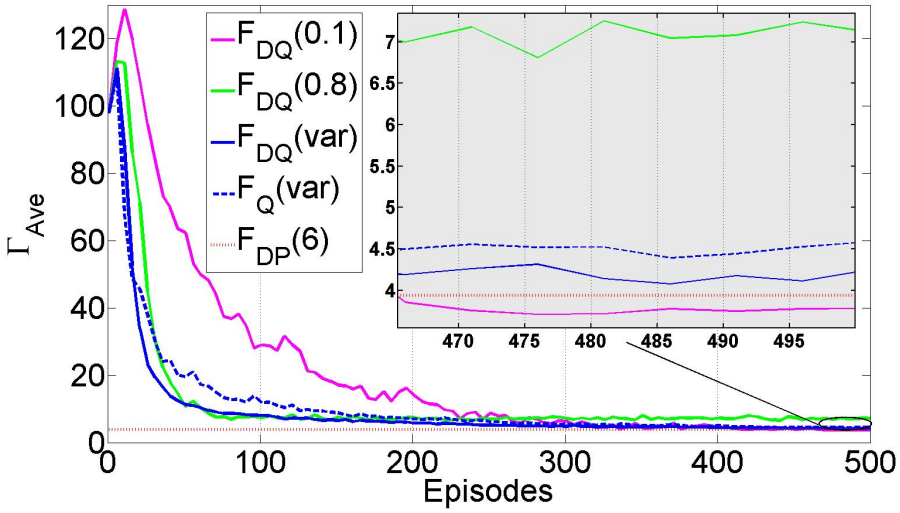

Figure 4.11: Learning curves for Dyna-Q-flocking in Exp. I

Exp. I: Dyna-Q-flocking

The learning curves for the policies learned through Dyna-Q-flocking are shown in Figure 4.11. According to the learning curves, the followers were able to learn how to flock with the leader using the Dyna-Q-flocking approach. As shown by Figure 4.12, the distributions of Γ for the learned policies are similar to $\Gamma(F_{DP}(6))$. Using the same set of hypotheses as Q-flocking but swapping F_Q for F_{DQ}, the t-test results shown in Table 4.5 reveal no significant differences between the distribution of $\Gamma(F_{DP}(6))$ and the distributions of both $\Gamma(F_{DQ}(0.1))$ and $\Gamma(F_{DQ}(var))$. On the other hand, the distribution of $\Gamma(F_{DQ}(0.8))$ is for certain (p-value=1) larger than $\Gamma(F_{DP}(6))$. These results confirm that the policies $F_{DQ}(var)$ and $F_{DQ}(0.1)$ are comparable to $F_{DP}(6)$ in performance, while $F_{DQ}(0.8)$ under-performs relative to $F_{DP}(6)$.

In order to compare Dyna-Q-flocking against Q-flocking, we used $F_Q(var)$ as the benchmark because it produced the best compromise between the rate of convergence and the lowest average cost incurred upon convergence. The learning curves plotted in Figure 4.11 show that $F_{DQ}(0.8)$ and $F_{DQ}(var)$ both converged slightly faster than $F_Q(var)$. However, between the two, only $F_{DQ}(var)$ converged to an average cost that is near $\Gamma_{Ave}(F_Q(var))$. According to Table 4.6, $F_{DQ}(0.1)$ and $F_{DQ}(var)$ converged to values that are within

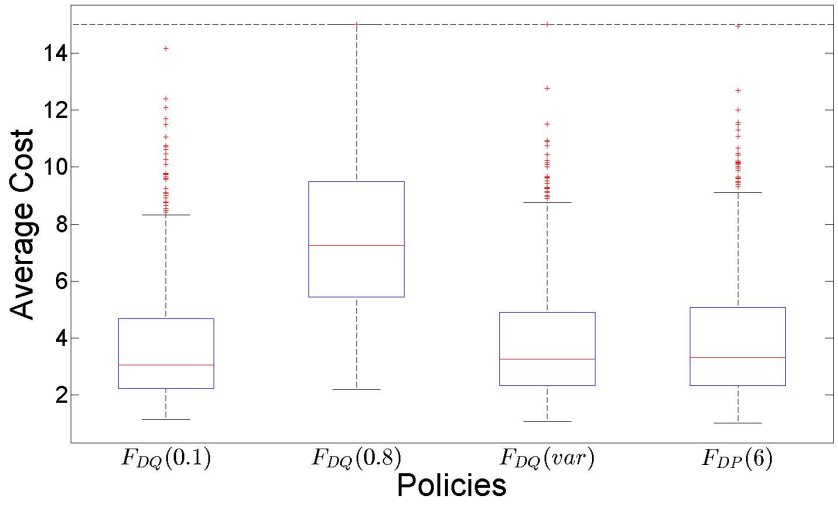

Figure 4.12: Box plot of Γ for all Dyna-Q-flocking policies in Exp. I

the vicinity of $\Gamma_{Ave}(F_Q(var))$. Based on these results, Dyna-Q-flocking and Q-flocking both augmented with the proposed variable learning parameter offers comparable performances in terms of convergence rate and value. As mentioned before, Q-flocking employs 50,000 followers on-line, while Dyna-Q-flocking employs five followers on-line and 50,000 followers in the planner. This difference makes Dyna-Q-flocking a more practical and realistic approach in terms of hardware implementation, because it requires less agents operating on-line.

To maximize the utility of the planner in Dyna-Q-flocking, we simulated Dyna-Q-flocking with 200,000 followers in the planner, and compared the results to Q-flocking and Dyna-Q-flocking with 50,000 followers in the planner. According to the learning curves shown in Figure 4.13, significant speed up was achieved with four times the original number of followers in the planner. As a side note, the Dyna-Q-flocking policies have the numbers 50 and 200 in their subscript to indicate the number of followers (multiplied by 1000) learning in the planner. Using $\Gamma_{Ave}(F_{DP}(6))$ as the cost benchmark, we can see from Figure 4.13 that $F_{DQ50}(var)$ converged below the benchmark after 465 episodes, whereas $F_{DQ200}(var)$ converged below the benchmark after 225. The improvement in the convergence rate due to scaling the number of agents

Table 4.5: T-test results for Dyna-Q-flocking in Exp. I

$H1_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_{DQ}(0.1)$	-2.425	1998	0.015	-0.224	2.065	-0.528	0.080
$F_{DQ}(0.8)$	30.883	1998	1.508E-171	3.924	2.841	3.505	4.343
$F_{DQ}(var)$	-0.988	1998	0.323	-0.091	2.056	-0.394	0.212

$H2_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_{DQ}(0.1)$	-2.425	1998	0.008	-0.224	2.065	-Inf	-0.072
$F_{DQ}(0.8)$	30.883	1998	1.000	3.924	2.841	-Inf	4.133
$F_{DQ}(var)$	-0.988	1998	0.162	-0.091	2.056	-Inf	0.060

Table 4.6: Γ_{Ave} of policies learned using Dyna-Q-flocking in Exp. I after 1000 episodes

Method	μ	σ
$F_{DQ}(0.1)$	3.712	2.023
$F_{DQ}(0.8)$	7.860	3.422
$F_{DQ}(var)$	3.663	2.068
$F_{Q}(var)$	3.809	2.224
$F_{DP}(6)$	3.936	2.106

Note: For F_{DQ}, the values were sampled after 1000 episodes. For F_{DP}, the values were sampled after the number of backups as indicated.

in the planner is one of the advantages of the Dyna architecture. The number of learning agents that can be simulated in the planner is limited by the computational resources available to run the planner in the background or in parallel to on-line learning. In comparing the performance between using different numbers of learning agents in the planner, we have shown that once an internal model of environment has been learned, the planner can be used

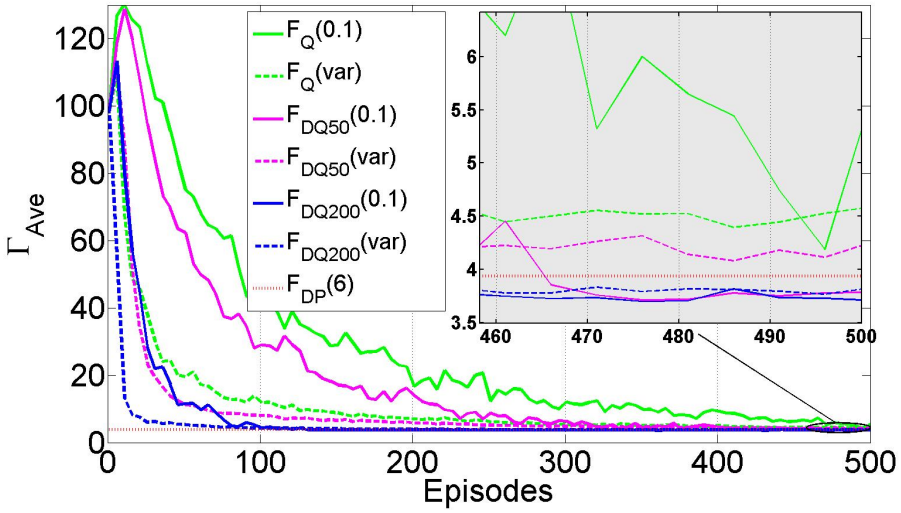

Figure 4.13: Learning curve for Q-flocking and Dyna-Q-flocking in Exp. I

as a force multiplier to expedite the learning process.

Exp. I: Summary

The results from Exp. I demonstrate that by using either Q-flocking or Dyna-Q-flocking, the followers were able to learn policies that facilitate flocking with a single leader, while operating in a simulated stochastic environment as defined by M_1. In addition, the t-test results confirmed that there are no significant differences between the costs incurred by $F_Q(0.1)$, $F_Q(var)$, $F_{DQ50}(0.1)$, $F_{DQ50}(var)$, $F_{DQ200}(0.1)$, and $F_{DQ200}(var)$ in comparison to $F_{DP}(6)$. This means that the learned policies mentioned have all converged to a near optimal policy, since according to the QDP approach, $F_{DP}(6)$ is the optimal control policy (lowest cost) for an environment defined by M_1. Consequently, we have shown empirically that policies learned using Q-flocking and Dyna-Q-flocking (along with low static α or the proposed variable α) can converge to a near optimal policy provided that enough time is allotted for the policies to converge. Lastly, the learning curves from Exp. I showed that in comparison to static learning parameters, the proposed variable learning parameter offers faster convergence during transitional phases without converging to policies that incur a higher cost (e.g., policies that used $\alpha = 0.8$).

4.3.2 Experiment II

The aim of Exp. II was to demonstrate that with the learning approaches the followers would be able to adapt their policies to the new environment defined by the model M_2. In addition, the learning agents in Exp. II were bootstrapped with a learned policy ($F_Q(0.1)$) from Exp. I. This transition from M_1 to M_2 simulated the perception of a non-stationary environment. In accordance with the QDP approach, which computes policies *off-line*, the F_{DP} policies from Exp. I remain unchanged in Exp. II even though the environment has. Consequently, followers relying on F_{DP} may incur a higher cost in an environment that F_{DP} was not optimize for. As a side note, the notation for the policies learned in Exp. II include ns in their subscript to differentiate them from the policies learned in Exp. I.

Similar to Exp. I, we simulated the learning approaches and tracked Γ_{Ave} for each of the policies as they evolved over every episode. Likewise, we evaluated the F_{DP} policies in the new environment M_2, where in addition to stochasticity in the roll angle dynamics and the airspeed, we introduced non-symmetrical noises into the model. The additional noises were used to simulate disturbances such as constant winds, or mechanical damage that occurs mid-flight, which may lead to weight and balance issues. Such disturbances can alter the flight characteristics of the UAV, as depicted by the followers in Figure 4.14. It should be noted that the additional disturbances were only applied to the followers, since it made more sense to show that the leader and follower would, in practice, experience different disturbances. This is the reason why in Figure 4.14, the leader exhibits a smooth trajectory, while the followers experience forces that pushes them in the positive x and y direction. In the same figure, the follower in green is shown to have executed a series of aggressive actions (at location: x=0, y=60) according to its policy $F_{Qns}(0.1)$, which facilitated better flocking with the leader compared to the follower in blue.

As shown in Figure 4.15, the F_{DP} policies generated between 40 to 60 backups seem to incur the lowest cost in the new environment. However, to remain in line with the QDP approach, $F_{DP}(6)$ was once again used as the benchmark in Exp. II. The mean and standard deviation of $\Gamma(F_{DP}(6))$ are shown in Table 4.7.

Exp. II: Q-flocking

According to the learning curves shown in Figure 4.16 and the convergence values compiled in Table 4.8, $\Gamma_{Ave}(F_{Qns}(0.1))$, $\Gamma_{Ave}(F_{Qns}(0.8))$ and $\Gamma_{Ave}(F_{Qns}(var))$

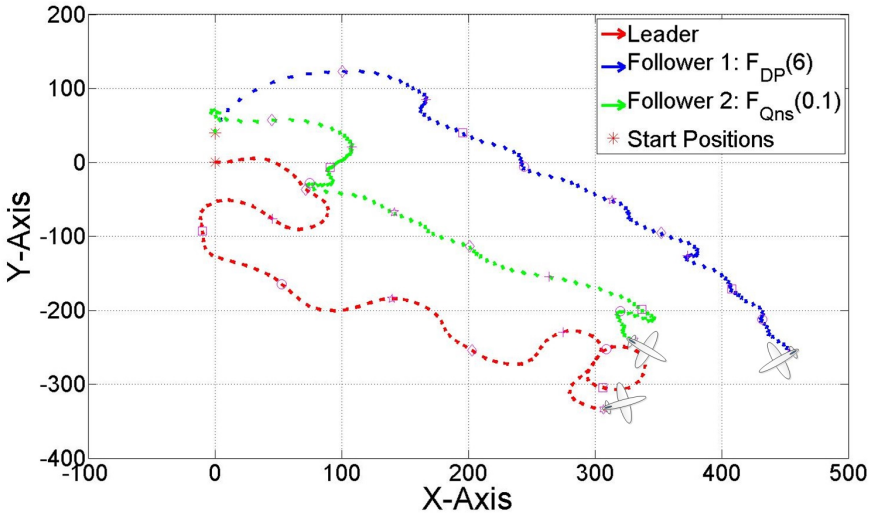

Figure 4.14: Trajectory plot of two followers flocking with a single leader in the environment defined by M_2. One follower is using $F_{Qns}(0.1)$ (after convergence), and the other is using $F_{DP}(6)$. For comparison purposes, the followers start int he same state, and markers are used to show the relative positions of each UAV in 10 second increments.

all converged to a lower value than $\Gamma_{Ave}(F_{DP}(6))$. This means that by using the Q-flocking approach, the followers were able to adapt their policies to M_2, and as a result, their policies incurred a lower average cost than $F_{DP}(6)$. Furthermore, the learning curves in Figure 4.16 show the policy with the proposed variable learning rate exhibiting faster convergence during the initial readjustment phase.

To validate Γ for all the Q-flocking policies in Exp. II, statistical significance testing was performed on Γ sampled after 1000 episodes using the same hypotheses as proposed in Exp. I. According to the box plots shown in Figure 4.17 and the t-test results compiled in Table 4.9, we can confirm with statistically strong evidence (i.e., $p < 0.001$) that the distributions of $\Gamma(F_{Qns}(0.1))$, $\Gamma(F_{Qns}(0.8))$, and $\Gamma(F_{Qns}(var))$ are significantly different from $\Gamma_{Ave}(F_{DP}(6))$, thus the learned policies must be different from $F_{DP}(6)$. In addition, there is significant evidence to confirm that all of the learned policies have incurred average costs that are lower in comparison to $F_{DP}(6)$.

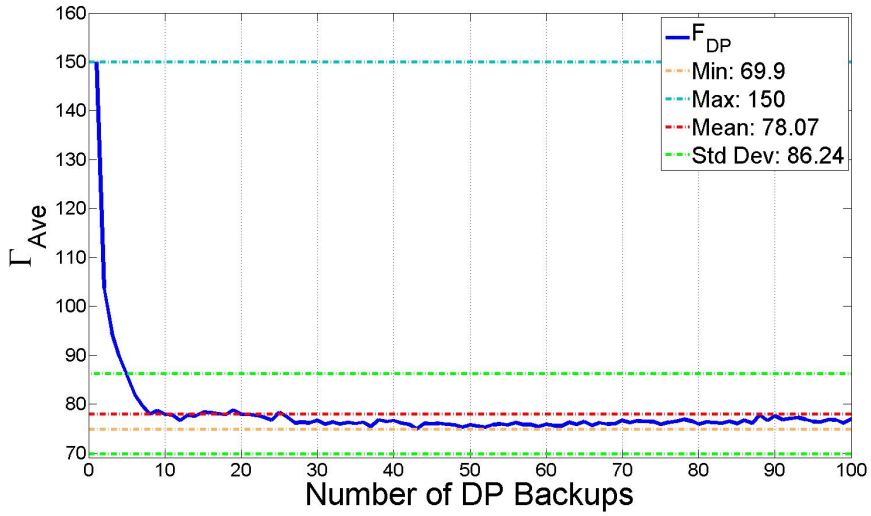

Figure 4.15: Plot of Γ_{Ave} for every DP backup using the QDP approach in Exp. II

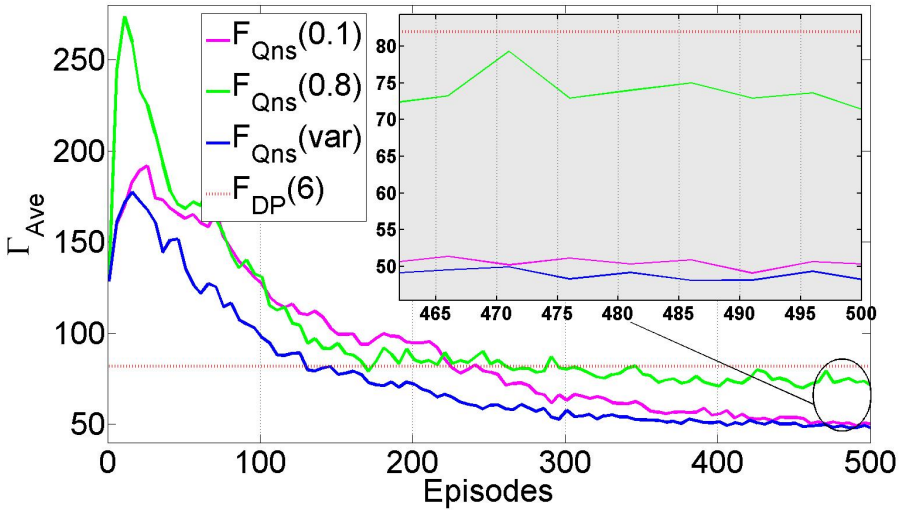

Figure 4.16: Learning curves for Q-flocking in Exp. II

Table 4.7: Γ_{Ave} of QDP policies in Exp. II

F_{DP}	μ	σ
1	150.0215	109.7961
2	103.7355	65.70306
3	94.7476	57.12971
4	89.7643	52.08801
5	85.9242	50.9022
6	82.0628	50.31499
7	79.7461	50.20371
8	77.9961	49.65202
9	78.7133	49.96488
10	77.9444	49.5283
⋮	⋮	⋮
100	77.0047	50.1035

Table 4.8: Γ_{Ave} of policies learned using Q-flocking in Exp. II after 1000 episodes

Method	μ	σ
$F_{Qns}(0.1)$	43.730	35.541
$F_{Qns}(0.8)$	72.500	66.724
$F_{Qns}(var)$	44.223	32.170
$F_{DP}(6)$	82.063	50.315

Note: For F_{Qns}, the values were sampled after 1000 episodes. For F_{DP}, the values were sampled after the number of backups as indicated.

Exp. II: Dyna-Q-flocking

The learning curves for the policies learned through Dyna-Q-flocking are shown in Figure 4.18. According to the learning curves, the followers were able to adapt their policies to M_2, but only $F_{DQns}(0.1)$ and $F_{DQns}(var)$ converged to costs that were lower than $\Gamma_{Ave}(F_{DP}(6))$ (see Table 4.10). The box plots of Γ for the learned policies (see Figure 4.19) show that the distributions

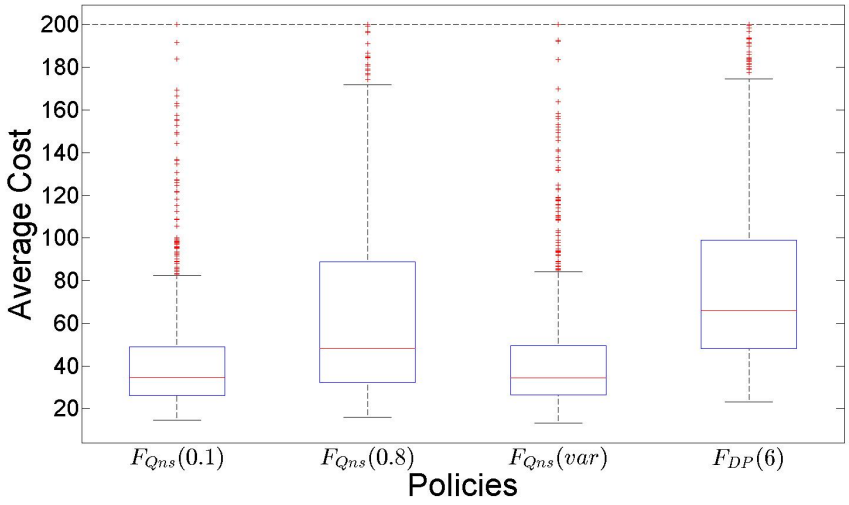

Figure 4.17: Distribution of Γ for all Q-flocking policies in Exp. II

of $\Gamma(F_{DQns}(0.1))$ and $\Gamma(F_{DQns}(var))$ are different from $\Gamma(F_{DP}(6))$, however, the distribution of $\Gamma(F_{DQns}(0.8))$ has similar means and considerable overlap with $\Gamma(F_{DP}(6))$. Using the same set of hypotheses as Q-flocking by swapping F_{Qns} for F_{DQns}, the t-test results shown in Table 4.11 confirms that $F_{DQns}(0.1)$ and $F_{DQns}(var)$ incurred lower average costs than $F_{DP}(6)$, while $F_{DQns}(0.8)$ incurred higher average costs than $F_{DP}(6)$.

In terms of comparing the learning approaches, the learning curves in Figure 4.18 show that $F_{Qns}(var)$ converge slightly faster than $F_{DQns}(var)$ and $F_{DQns}(0.1)$ in the first 200 episodes, after which the three policies seem to converge at the same rate and to within the vicinity of the same value. Based on this observation, there is no clear indication as to which learning approach performs better when the simulation conditions are similar (e.g., in Q-flocking 50,000 followers were simulated learning on-line versus in Dyna-Q-flocking 50,000 followers were simulated in the planner). This is consistent with the findings in Exp. I.

As mentioned before, the advantage of the Dyna architecture lies in the ability to expedite the learning process through learning internal models and scaling the number of learners with the planner. For this reason, we also simulated Dyna-Q-flocking with 200,000 followers in the planner. Using $\Gamma_{Ave}(F_{DP}(6))$

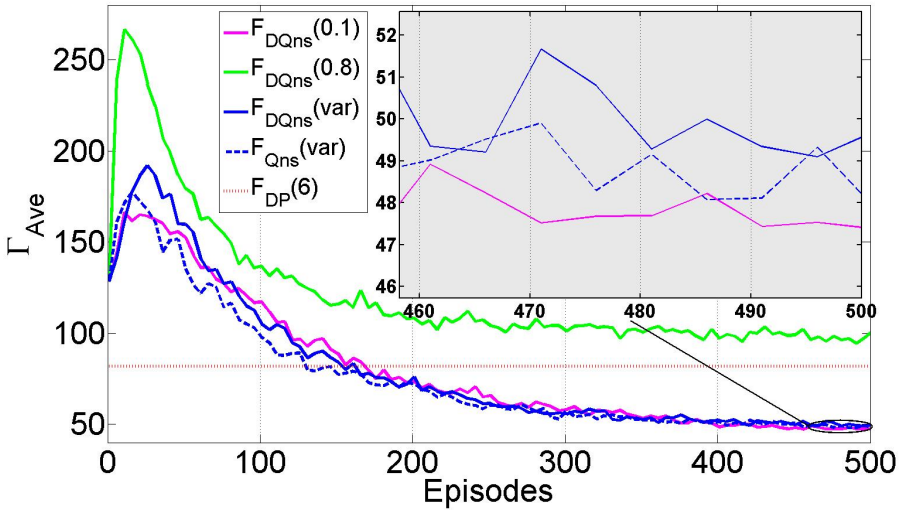

Figure 4.18: Learning curves for Dyna-Q-flocking in Exp. II

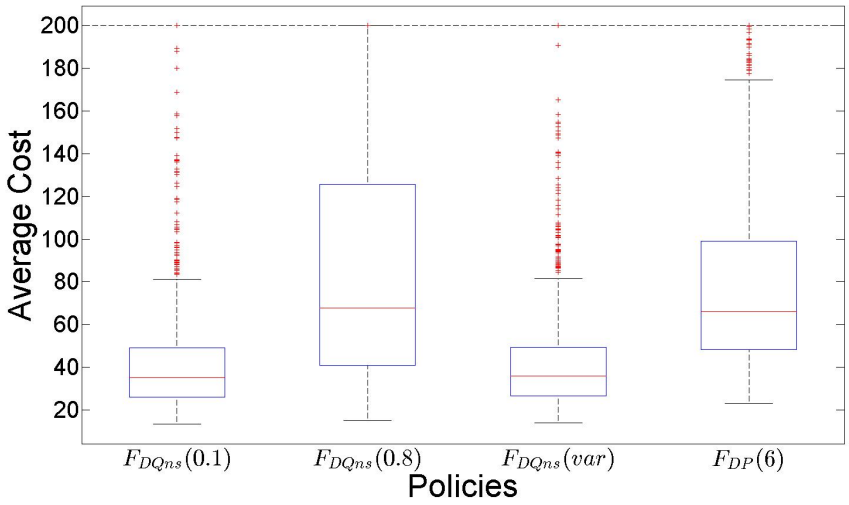

Figure 4.19: Distribution of Γ for all Dyna-Q-flocking policies in Exp. II

Table 4.9: T-test results for Q-flocking in Exp. II

$H1_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_{Qns}(0.1)$	-19.678	1998	6.165E-79	-38.333	43.559	-44.752	-31.913
$F_{Qns}(0.8)$	-3.619	1998	3.037E-04	-9.563	59.092	-18.271	-0.854
$F_{Qns}(var)$	-20.037	1998	1.577E-81	-37.840	42.228	-44.063	-31.616

$H2_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_{Qns}(0.1)$	-19.678	1998	3.082E-79	-38.333	43.559	-Inf	-35.127
$F_{Qns}(0.8)$	-3.619	1998	1.518E-04	-9.563	59.092	-Inf	-5.214
$F_{Qns}(var)$	-20.037	1998	7.887E-82	-37.840	42.228	-Inf	-34.732

Table 4.10: Γ_{Ave} of policies learned using Dyna-Q-flocking in Exp. II after 1000 episodes

Method	μ	σ
$F_{DQns}(0.1)$	42.607	28.190
$F_{DQns}(0.8)$	95.097	73.270
$F_{DQns}(var)$	43.398	28.000
$F_{DP}(6)$	3.936	2.106

Note: For F_{DQns}, the values were sampled after 1000 episodes. For F_{DP}, the values were sampled after the number of backups as indicated.

as the cost benchmark, there is a reduction of 125 episodes between $F_{DQ50ns}(0.1)$ and $F_{DQ200ns}(0.1)$ as shown in Figure 4.20. Furthermore, in comparison to $F_{Qns}(0.1)$, $F_{DQns200}(0.1)$ reduced the number of episodes by nearly 200. These improvements are consistent with the findings from Exp. I, thus attests to the efficacy of integrating model learning and planning with on-line learning.

Table 4.11: T-test results for Dyna-Q-flocking in Exp. II

$H1_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_{DQns}(0.1)$	-21.633	1998	2.014E-93	-39.455	40.782	-45.466	-33.445
$F_{DQns}(0.8)$	4.637	1998	3.755E-06	13.035	62.849	3.772	22.297
$F_{DQns}(var)$	-21.234	1998	2.159E-90	-38.664	40.716	-44.665	-32.664

$H2_0$

				Paired Differences			
						99.9% Confidence Interval	
Policy	T-values	DoF	Significance	Mean	Std. Dev.	Lower	Upper
$F_{DQns}(0.1)$	-21.633	1998	1.007E-93	-39.455	40.782	-Inf	-36.454
$F_{DQns}(0.8)$	4.637	1998	1.000E+00	13.035	62.849	-Inf	17.660
$F_{DQns}(var)$	-21.234	1998	1.079E-90	-38.664	40.716	-Inf	-35.668

Exp. II: Summary

The results from Exp. II demonstrate the feasibility of using either Q-flocking or Dyna-Q-flocking to enable the followers to adapt their policies to the new environment (i.e., M_2), hence validating the ability to adapt to a non-stationary stochastic environment. Policies that were learned using $\alpha = 0.1$ and $\alpha = var$ incurred lower average costs in comparison to $F_{DP}(6)$. Furthermore, the results show that for Q-flocking the proposed variable learning parameter offers faster convergence during readjustment phases without converging to policies that incur a higher cost (e.g., policies that used $\alpha = 0.8$).

4.4 Summary

In this chapter, we presented the simulation process, evaluation procedure, and simulation results for Q-flocking and Dyna-Q-flocking. The simulations occurred in an episodic format, such that in each episode multiple followers were learning how to flock with a single leader by updating a shared Q-table with estimates of the state-action values. Evaluation of the policies were based on the average cost incurred by the follower that utilize the policies while flocking with the leader over 1000 random but controlled trajectories. The distributions of the average costs were compared by performing two sample t-tests.

Two sets of experiments were conducted to simulate both Q-flocking and

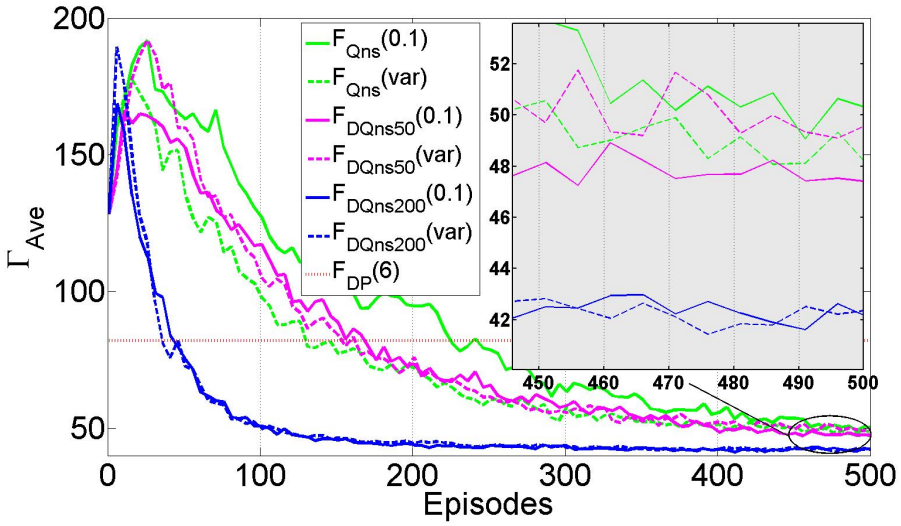

Figure 4.20: Learning curve for Q-flocking and Dyna-Q-flocking in Exp. II

Dyna-Q-flocking. The difference between the experiments was in the stochastic kinematic model of the UAV. By changing the parameters within the kinematic model, we were able to simulate the perception of a non-stationary stochastic environment.

Simulation results show that both learning approaches can be used to enable small fixed-wing UAVs to learn how to flock with a leader without a priori knowledge of the environment (See Figure 4.21 and 4.22). In addition, the learned policies with low or variable learning parameters exhibited comparable performance to the policy generated using the QDP approach. Furthermore, the results show that learning provided the followers with the means of adapting to new environments (See Figure 4.23 and 4.24) by constantly exploring the environment, and readjusting their state-action values. We were able to test the proposed variable learning parameter in both environments, and show that Q-flocking combined with the variable learning rates improves the speed of convergence, most notably during the initial learning and readjustment phases of the learning process.

In comparison to Q-flocking, Dyna-Q-flocking eliminated the need for a large number of on-line agents. Sample efficiency was improved through learning a model of the environment, and using the planner to generate simulated experience, which were then used to update the Q-table. By removing the need

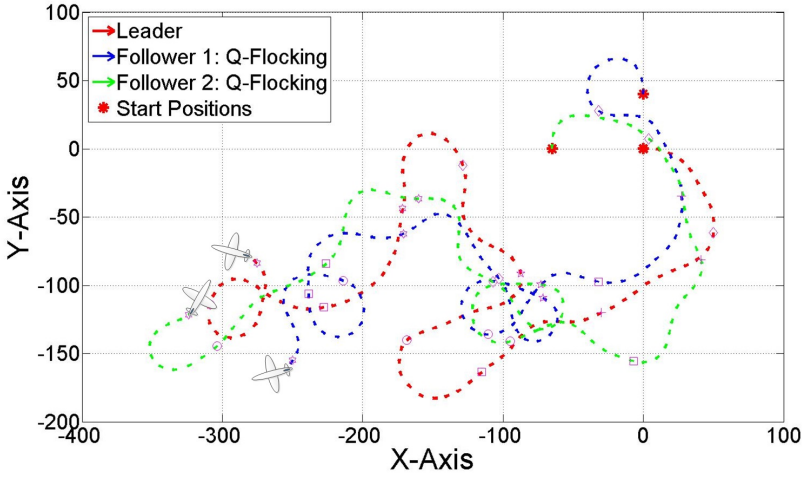

Figure 4.21: Trajectories of two followers using $F_Q(0.1)$ to flock with a single leader in the environment defined by M_1. Markers are shown in 10 second increments.

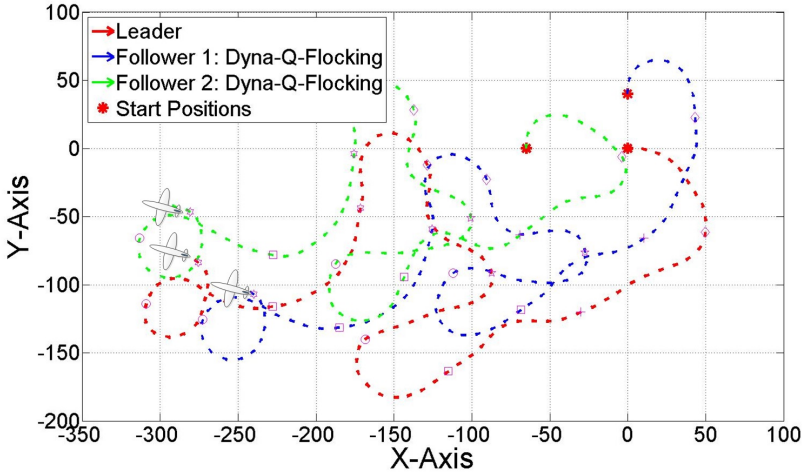

Figure 4.22: Trajectories of two followers using $F_{DQ}(0.1)$ to flock with a single leader in the environment defined by M_1. Markers are shown in 10 second increments.

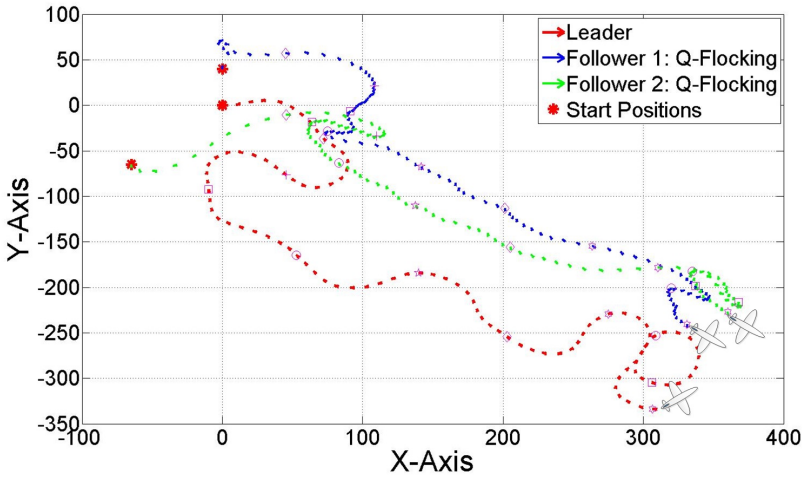

Figure 4.23: Trajectories of two followers using $F_{Qns}(0.1)$ to flock with a single leader in the environment defined by M_2. Markers are shown in 10 second increments.

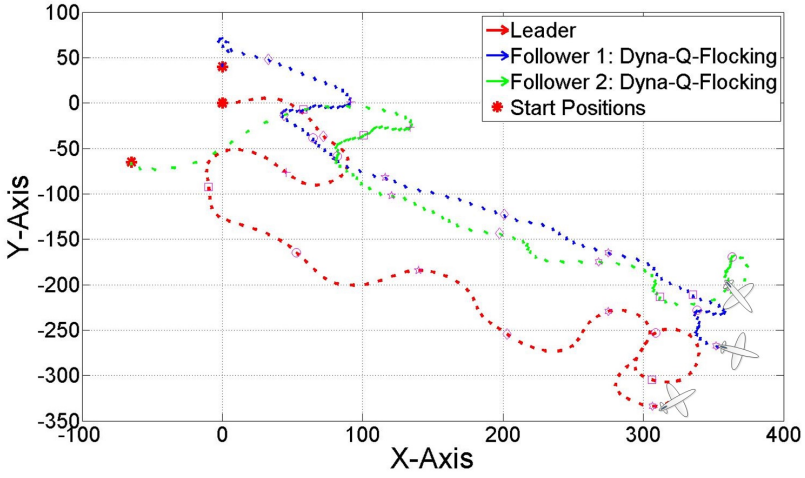

Figure 4.24: Trajectories of two followers using $F_{DQns}(0.1)$ to flock with a single leader in the environment defined by M_2. Markers are shown in 10 second increments.

of having to use 50,000 followers to learn on-line, and by scaling up the number of learners simulated in the planner, we have shown that Dyna-Q-flocking is a better practical approach to the proposed flocking problem, however, several assumptions still need to be addressed. These include an episodic setting where agents can be reset if they moved out of bounds, lossless communication between the agents, and slow changing disturbances to accommodate time factor of learning a policy (due to the size of state-space). Lastly, flocking was only achievable because collision avoidance was de-conflicted by having each UAV flying at different altitudes. These are some of the assumptions that need to be resolved prior to hardware implementation.

5 Conclusion and Recommendations

5.1 Conclusion

The aim of this research was to apply RL to flocking so that agents modeled as small fixed-wing UAV can learn how to flock in a simulated non-stationary stochastic environment. We adopted the flocking framework from [14], and formulated it in the context of an RL problem. The state space was defined in terms of the relationship between a follower and the leader, and the action space was defined by a set of discretized roll angle setpoints. In addition, the state transitions were described by a four DoF stochastic UAV kinematic model and the reinforcement was formed as a cost function that integrated the rules of flocking. The objective was for the followers to learn the best roll command to take in all the states.

To address the RL problem, we proposed two learning approaches, namely Q-flocking and Dyna-Q-Flocking. The algorithm in the Q-flocking approach is based on Peng's Q(λ) augmented with a variable learning parameter, while the algorithm in Dyna-Q-flocking is based on Sutton's Dyna architecture and Q(λ).

Two sets of simulation experiments were carried out in this research. The aim of the first experiment was to learn a policy that facilitated flocking. In the second experiment, the aim was to adapt the learned policy to a new environment. By changing the parameters within the kinematic model, we were able to simulate different environments and create the perception of a non-stationary stochastic environment. The simulations occurred in an episodic format, such that in each episode the followers learned how to flock with a single leader using the proposed learning algorithms. Evaluation of the learned policies was done by comparing the average cost (in terms of the cost function) incurred over a controlled set of 1000 random trajectories, where the follower

would use the policies to flock with the leader. Statistical analysis was performed on the average cost data sets from each policy in order to compare the learning approaches to the QDP approach.

According to the simulation results, both learning approaches can be used by small fixed-wing UAVs to learn how to flock with a leader in a simulated non-stationary stochastic environment. Furthermore, the results show that under the same simulation conditions, the policies learned with a low static α or the proposed variable α exhibited comparable performance to the optimal control policy generated using the QDP approach. With the Dyna-Q approach, we were able to reduce the number of on-line learners to a manageable level, and scale up the number of learning agents in the planner, such that the policies converged faster than using the Q-flocking approach.

Based on the experiments and results, we have demonstrated the feasibility of learning how to flock using fixed-wing UAVs in a simulated environment. However, several assumptions discussed at the end of Chapter 4 need to be relaxed and resolved prior to hardware implementation.

5.2 Contributions

As UAVs become more and more prevalent in military and industrial applications, there is a drive towards cooperative multi-UAV systems in order to benefit from economies of scale. For such systems to operate effectively, individual agents will require more than preprogrammed rules and good controllers; they will need tools for learning how to execute complex tasks and learn to adapt to unfamiliar situations.

The integration of RL and flocking with fixed-wing UAVs in a simulated non-stationary stochastic environment is a step towards developing intelligent agents that can learn to flock in the physical world.

In summary, this thesis has contributed to the design of intelligent agents by introducing an RL flocking framework for fixed-wing UAVs and in support of this contribution:

- Adopted a stochastic optimal control problem from [14] and reformulated it in the context of an RL problem.
- Formulated two learning approaches called Q-flocking and Dyna-Q-flocking. The former is based on Peng's $Q(\lambda)$, and the latter is a combination of Peng's $Q(\lambda)$ and Sutton's Dyna architecture.
- Demonstrated in simulation a model-free RL approach to flocking with small fixed-wing UAVs in a non-stationary stochastic environment by applying Q-flocking.

- Demonstrated in simulation a Dyna-Q approach to simulated flocking with small fixed-wing UAVs in a non-stationary stochastic environment, where the UAVs are assumed to be flocking in a leader-follower topology.
- A comparison study of each approach to Quintero's DP approach.

5.3 Recommendations for Future Work

The following are ideas for future work:

1. Reducing state-space: To reduce the state-space, the first two sub-states of the system state can be converted to polar coordinates. This will allow discretization of the distance to the leader using less states. However, the resolution of each state on the planar surface that is traveled on will differ; the closer the follower is to the leader the higher the resolution, and vice versa.
2. Improve Model Learning: Instead of averaging the samples to create a model, a collection of the samples or at least a set of the most recent samples should be stored. This will provide more realistic samples for the planner to work with.
3. Parallel computation: Currently, on-line learning, model learning, and planning are running sequentially. If each process runs on a separate processor similar to what is proposed in [58], the planner would be able to update more states and at a faster rate. More importantly, other than reducing the state space, parallelization is the only way learning can be fast enough to meet real-time constraints.
4. Policy Search: We have not looked into policy search methods of learning, but we suspect that with the symmetry of the problem in its current formulation, policy search may be used to eliminate the large Q-table.
5. Collision avoidance: Collision avoidance is still an open issue that needs to be resolved either at a lower level controller (see [4]), through the cost function, or through additional states (not ideal).
6. Flock geometries: This was mentioned by Quintero, which is to induce certain flock geometries through the cost function.

Bibliography

[1] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 25–34.

[2] M. Dewi, M. Hariadi, and M. H. Purnomo, "Simulating the movement of the crowd in an environment using flocking," in *Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), 2011 2nd International Conference on*, 2011, pp. 186–191.

[3] G. Colqui, M. Tomita, T. Hattori, and Y. Chigusa, "New video synthesis based on flocking behavior simulation," in *Communications, Control and Signal Processing, 2008. ISCCSP 2008. 3rd International Symposium on*, 2008, pp. 936–941.

[4] H. M. La and W. Sheng, "Distributed sensor fusion for scalar field mapping using mobile sensor networks," *Cybernetics, IEEE Transactions on*, vol. 43, no. 2, pp. 766–778, April 2013.

[5] S. Semnani and O. Basir, "Semi-flocking algorithm for motion control of mobile sensors in large-scale surveillance systems," *Cybernetics, IEEE Transactions on*, vol. 45, no. 1, pp. 129–137, Jan 2015.

[6] Y. U. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997.

[7] Y. Tan and Z. yang Zheng, "Research advance in swarm robotics," *Defence Technology*, vol. 9, no. 1, pp. 18–39, 2013.

[8] A. E. Turgut, H. Celikkanat, F. Gokce, and E. Sahin, "Self-organized flocking with a mobile robot swarm," in *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 39–46.

[9] H. La and W. Sheng, "Flocking control algorithms for multiple agents in cluttered and noisy environments," in *Bio-Inspired Self-Organizing Robotic Systems*. Springer Berlin Heidelberg, 2011, vol. 355, pp. 53–79.

[10] B. Lerner, "UAVs and force: Current debates and future trends in technology, policy and the law," *Center for Security Policy Occasional Paper Series*, 2013.

[11] K. Anderson, "Rise of the drones: unmanned systems and the future of war," *Hearings before the Subcommittee on National Security and Foreign Affairs, 111th Cong., 2d Sess.*, 2010.

[12] G. Vásárhelyi, C. Virágh, G. Somorjai, N. Tarcai, T. Szörényi, T. Nepusz, and T. Vicsek, "Outdoor flocking and formation flight with autonomous aerial robots," *arXiv preprint arXiv:1402.3588*, 2014.

[13] O. Saif, I. Fantoni, and A. Zavala-Rio, "Flocking of multiple unmanned aerial vehicles by lqr control," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, May 2014, pp. 222–228.

[14] S. A. P. Quintero, G. E. Collins, and J. P. Hespanha, "Flocking with fixed-wing UAVs for distributed sensing: A stochastic optimal control approach," in *American Control Conference (ACC), 2013*, 2013, pp. 2025–2031, iD: 1.

[15] A. Chapman and M. Mesbahi, "Uav flocking with wind gusts: Adaptive topology and model reduction," in *American Control Conference (ACC), 2011*, June 2011, pp. 1045–1050.

[16] R. Olfati-saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Transactions on Automatic Control*, vol. 51, pp. 401–420, 2006.

[17] C. Watkins, "Learning from delayed rewards," PhD Dissertation, University of Cambridge, 1989.

[18] M. Tomimasu, K. Morihiro, H. Nishimura, T. Isokawa, and N. Matsui, "A reinforcement learning scheme of adaptive flocking behavior," in *Proc. of the 10th Int. Symp. on Artificial Life and Robotics (AROB)*, 2005.

[19] K. Morihiro, T. Isokawa, H. Nishimura, and N. Matsui, "Characteristics of flocking behavior model by reinforcement learning scheme," in *SICE-ICASE, 2006. International Joint Conference*, Oct 2006, pp. 4551–4556.

[20] H. M. La, R. Lim, and W. Sheng, "Multirobot cooperative learning for predator avoidance," *Control Systems Technology, IEEE Transactions on*, vol. 23, no. 1, pp. 52–63, Jan 2015.

[21] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," *Machine Learning*, vol. 22, pp. 283–290, 1996.

[22] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *In Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, 1990, pp. 216–224.

[23] J. P. How, C. Fraser, K. C. Kulling, L. F. Bertuccelli, O. Toupet, L. Brunet, A. Bachrach, and N. Roy, "Increasing autonomy of UAVs," *Robotics and Automation Magazine, IEEE*, vol. 16, no. 2, pp. 43–51, 2009.

[24] X. Liu, P. Chen, X. Tong, S. Liu, S. Liu, Z. Hong, L. Li, and K. Luan, "UAV-based low-altitude aerial photogrammetric application in mine areas measurement," in *Earth Observation and Remote Sensing Applications (EORSA), 2012 Second International Workshop on*, June 2012, pp. 240–242.

[25] E. Hunt, C. Daughtry, S. Mirsky, and W. Hively, "Remote sensing with unmanned aircraft systems for precision agriculture applications," in *Agro-Geoinformatics (Agro-Geoinformatics), 2013 Second International Conference on*, Aug 2013, pp. 131–134.

[26] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic uav and ugv system for precision agriculture," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 5321–5326.

[27] E. Adamey and U. Ozguner, "A decentralized approach for multi-UAV multitarget tracking and surveillance," *Proc. SPIE, Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR III*, vol. 8389, p. 15, 2012.

[28] T. Shima and S. Rasmussen, *UAV Cooperative Decision and Control: Challenges and Practical Approaches*. USA: Society for Industrial and Applied Mathematics, 01/01; 2014/02 2009.

[29] L. Parker, "Multiple mobile robot teams, path planning and motion coordination," in *Encyclopedia of Complexity and Systems Science*, R. A. Meyers, Ed. Springer New York, 2009, pp. 5783–5800.

[30] E. L. Hayes, "Machine learning for intelligent control: Application of reinforcement learning techniques to the development of flight control systems for miniature uav rotorcraft," 2013.

[31] M. Grana, B. Fernandez-Gauna, and J. M. Lopez-Guede, "Cooperative multi-agent reinforcement learning for multi-component robotic systems: guidelines for future research," *Paladyn*, vol. 2, no. 2, pp. 71–81, 2011.

[32] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[33] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, pp. 237–285, 1996.

[34] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[35] H. M. Schwartz, *Multi-Agent Reinforcement Learning.* USA: John Wiley and Sons, Inc., 2014.

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

[37] A. Gosavi, "Reinforcement learning: A tutorial survey and recent advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, 2009.

[38] M. van Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement Learning: State of the Art*, ser. Adaptation, Learning, and Optimization. Springer, 2012, vol. 12, pp. 3–42.

[39] P. Kormushev, S. Calinon, and D. G. Caldwell, "Reinforcement learning in robotics: Applications and real-world challenges," *Robotics*, vol. 2, no. 3, pp. 122–148, 2013.

[40] L. Buşoniu, R. Babuška, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 2, pp. 156–172, March 2008.

[41] R. E. Bellman, *Dynamic Programming.* Princeton New Jersey: Princeton University Press, 1957.

[42] T. M. Moldovan and P. Abbeel, "Safe exploration in markov decision processes," in *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, UK, 2012, 1205.4810.

[43] A. D. Laud, "Theory and application of reward shaping in reinforcement learning," PhD Dissertation, University of Illinois at Urbana-Champaign, 2004.

[44] M. L. Littman, "Value-function reinforcement learning in markov games," *Cognitive Systems Research*, vol. 2, no. 1, pp. 55–66, 2001.

[45] M. Lauer and M. Riedmiller, "An algorithm for distributed reinforcement learning in cooperative multi-agent systems," in *In Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann, 2000, pp. 535–542.

[46] H. G. Tanner, A. Jadbabaie, and G. J. Pappas, "Stable flocking of mobile agents, Part I: Fixed topology," in *IEEE Conference on Decision and Control*, vol. 2, Dec 2003, pp. 2010–2015.

[47] ——, "Stable flocking of mobile agents, Part II: Dynamic topology," in *IEEE Conference on Decision and Control*, 2003, pp. 2016–2021.

[48] N. Moshtagh and A. Jadbabaie, "Distributed geodesic control laws for flocking of nonholonomic agents," *Automatic Control, IEEE Transactions on*, vol. 52, no. 4, pp. 681–686, April 2007.

[49] M. J. Mataric, "Interaction and intelligent behavior," PhD Dissertation, MIT, 1994.

[50] A. Hayes and P. Dormiani-Tabatabaei, "Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots," in *Robotics and Automation, 2002. Proceedings. ICRA 2002. IEEE International Conference on*, vol. 4, 2002, pp. 3900–3905.

[51] A. Campo, S. Nouyan, M. Birattari, R. Grob, and M. Dorigo, "Negotiation of goal direction for cooperative transport," in *Ant Colony Optimization and Swarm Intelligence*, ser. Lecture Notes in Computer Science, M. Dorigo, L. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stutzle, Eds. Springer Berlin Heidelberg, 2006, vol. 4150, pp. 191–202.

[52] J. Welsby and C. Melhuish, "Autonomous minimalist following in three dimensions: a study with small-scale dirigibles," *Proceedings of Towards Intelligent Mobile Robots*, 2001.

[53] S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J. C. Zufferey, and D. Floreano, "Reynolds flocking in reality with fixed-wing robots: Communication range vs. maximum turning rate," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011, pp. 5015–5020.

[54] C. Virágh, G. Vásárhelyi, N. Tarcai, T. Szörényi, G. Somorjai, T. Nepusz, and T. Vicsek, "Flocking algorithm for autonomous flying robots," *Bioinspiration & Biomimetics*, vol. 9, no. 2, p. 025012, 2014.

[55] R. W. Beard and T. W. McLain, *Small unmanned aircraft: Theory and practice*. Princeton University Press, 2012.

[56] T. M. Foster and W. J. Bowman, "Dynamic stability and handling qualities of small unmanned-aerial-vehicles," Ph.D. dissertation, Brigham Young University. Department of Mechanical Engineering, 2005.

[57] W. Dabney and A. G. Barto, "Adaptive step-size for online temporal difference learning." in *Artificial Intelligence, 26th AAAI Conference on*, 2012.

[58] T. Hester, M. Quinlan, and P. Stone, "RTMBA a real-time model-based reinforcement learning architecture for robot control," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* IEEE, 2012, pp. 85–90.