Reinforcement Learning for Improved Utility of Simulation-Based Training

Johan Källström
Reinforcement Learning for Improved Utility of Simulation-Based Training

Johan Källström
ABSTRACT

Team training in complex domains often requires a substantial number of resources, e.g., vehicles, machines, and role-players. For this reason, it may be difficult to realise efficient and effective training scenarios in a real-world setting. Instead, part of the training can be conducted in synthetic, computer-generated environments. In these environments trainees can operate simulators instead of real vehicles, while synthetic actors can replace human role-players to increase the complexity of the simulated scenario at low operating cost. However, constructing behaviour models for synthetic actors is challenging, especially for the end users, who typically do not have expertise in artificial intelligence. In this dissertation, we study how machine learning can be used to simplify the construction of intelligent agents for simulation-based training. A simulation-based air combat training system is used as case study.

The contributions of the dissertation are divided into two parts. The first part aims at improving the understanding of reinforcement learning in the domain of simulation-based training. First, a user-study is conducted to identify important capabilities and characteristics of learning agents that are intended to support training of fighter pilots. It is identified that one of the most important capabilities of learning agents in the context of simulation-based training is that their behaviour can be adapted to different phases of training, as well as to the training needs of individual human trainees. Second, methods for learning how to coordinate with other agents are studied in simplified training scenarios, to investigate how the design of the agent’s observation space, action space, and reward signal affects the performance of learning. It is identified that temporal abstractions and hierarchical reinforcement learning can improve the efficiency of learning, while also providing support for modelling of doctrinal behaviour. In more complex settings, curriculum learning and related methods are expected to help find novel tactics even when sparse, abstract reward signals are used. Third, based on the results from the user study and the practical experiments, a system concept for a user-adaptive training system is developed to support further research.

The second part of the contributions focuses on methods for utility-based multi-objective reinforcement learning, which incorporates knowledge of the user’s utility function in the search for policies that balance multiple conflicting objectives. Two new agents for multi-objective reinforcement learning are proposed: the Tunable Actor (T-Actor) and the Multi-Objective Dreamer (MO-Dreamer). T-Actor provides decision support to instructors by learning a set of Pareto optimal policies, represented by a single neural network conditioned on objective preferences. This enables tuning of the agent’s behaviour to fit trainees’ current training needs. Experimental evaluations in gridworlds and in the target system show that T-Actor reduces the number of training steps required for learning. MO-Dreamer adapts online to changes in users’ utility, e.g., changes in training needs. It does so by learning a model of the environment, which it can use for anticipatory rollouts with a diverse set of utility functions to explore which policy to follow to optimise the return for a given set of objective preferences. An experimental evaluation shows that MO-Dreamer outperforms prior model-free approaches in terms of experienced regret, for frequent as well as sparse changes in utility.

Overall, the research conducted in this dissertation contributes to improved knowledge about how to apply machine learning methods to construction of simulation-based training environments. While our focus was on air combat training, the results are general enough to be applicable in other domains.
This work was partially supported by the Swedish Governmental Agency for Innovation Systems (grant NFFP7/2017-04885), and the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The computations were enabled by the resources provided by the Swedish National Infrastructure for Computing (SNIC) at Ttetralith/NSC partially funded by the Swedish Research Council through grant agreement no. 2020/5-230, as well as the supercomputing resource Berzelius provided by the National Supercomputer Centre at Linköping University and the Knut and Alice Wallenberg foundation.
Träning inom komplexa domäner kan kräva såväl materiella som mänskliga resurser i stor omfattning, till exempel fordon, maskiner och rollspelare som kan populera träningsscenari- on. Detta kan göra det svårt att genomföra träningen i den verkliga operativa miljön. Istället kan en del av träningen genomföras i syntetiska, datogenererade miljöer. I dessa miljöer kan de som tränas hantera simulatorer istället för riktiga fordon och maskiner, samtidigt som de interagerar med syntetiska aktörer istället för med mänskliga rollspelare. Detta kan göra det möjligt att realisera komplexa träningsscenarion till sänkt kostnad, men att konstruera realistiska beteendemodeller för de syntetiska aktörerna är en utmaning. Det gäller särskilt för träningssystemens användare, som ofta inte har de nödvändiga kunskaperna inom artificiell intelligens.


Avhandlingens bidrag presenteras i två delar. I den första delen fokuserar vi på att lära oss mer om domänegenskaper och användarbehov inom simuleringsbaserad pilotträning. Först genomförs en användarstudie bestående av intervjuer och enkäter, för att identifiera krav på datorstyrdas agenter som används för att stötta pilotträning. En viktig förmåga som identifieras är att kunna anpassa agenternas egenskaper till olika faser i utbildningen, samt till varje individ aktuella förmågenivå. Detta skiljer sig från många andra använd- ningsfall för förstärkningsinlärning, där man ofta eftersträvar en för människor oslagbar agent, som dessutom ofta har ett statiskt beteende. Därefter genomförs ett antal praktiska experiment med scenarion relaterade till koordination mellan flera lärande agenter i förenklade luftstridsscenarion, med avsikt att studera hur prestanda relaterar till design av agentens observationsrymd, handlingsrymd och belöningssignal. Experimenten indikerar att tidsabstraktioner och hierarkiskt beslutsfattande kan göra inlärningen mer effektiv och även underlätta modellering av beteende som i delar följer en känd doktrin. För att kunna lära i mer komplexa scenarion, så bedöms det vara lämpligt att använda adaptiva miljöer, där miljöns utmaning successivt ökas medan agenten lär sig. Slutligen sammanfattas resul- taten i form av en arkitektur för ett adaptivt träningssystem, där agenter med adapterbart beteende deltar som syntetiska aktörer i träningsscenarion, medan andra agenter anpassar innehållet i träningsscenarien baserat på hur piloternas förmåga utvecklas över tid.


T-Actor tillhandahåller beslutsstöd åt de instruktörer som bygger simuleringar för tränning, genom att optimera sitt agerande för många olika nyttofunktioner. Agentens policy för beslutsfattande realiseras av ett djupt neuronät, som utöver observationer av miljön även får den aktuella nyttofunktionen som indata. Genom att justera nyttofunktionen under simulering, så kan instruktören observera hur agentens beteende förändras och välja det som bäst passar för det aktuella träningsscenariot. Experiment i miljöer av varierande komplexitet visar att man genom att låta en agent utforska många nyttofunktioner kan förbättra effektiviteten jämfört med att träna flera olika agenter med statiska nyttofunktioner. MO-Dreamer anpassar sitt beteende i operativa system baserat på förändringar i användarens nyttofunktion, till exempel förändringar i studentpiloters träningsbehov. För att effektivisera anpassningen, så lär sig agenten en modell av miljön, som den kan använda för utforsknings simuleringar för att förbereda sig inför framtida förändringar i användares behov. En experimentell utvärdering visar att MO-Dreamer överträffar tidigare modellfria metoder för frekventa såväl som glesa förändringar i användares nyttofunktioner.

I sin helhet bidrar avhandlingen till ökade kunskaper kring användning av maskininlärning för konstruktion av simuleringsbaserade träningsmiljöer. Även om vårt fokus i arbetet har varit tränning av stridspiloter, så är resultaten av mer generell natur och kan användas i andra typer av träningssystem. Där kan agenter delta som syntetiska aktörer i komplexa simuleringsscenarion, hjälpa till att anpassa innehållet i träningsscenarion och stötta instruktörer i utvärdering av mänskliga deltagares prestationer och identifiering av framtida träningsbehov.
I would first like to thank the people that participated in this project. Thanks to my supervisors Fredrik Heintz at Linköping University and Erik Herzog at Saab Aeronautics for their support throughout the project, and thanks to Rego Granlund at RISE for helping out with the user study conducted in this project. Thanks also go to Jörgen Degerstedt for helping me initialise and manage this project. In addition to those directly involved in the project, I am also grateful to anyone that provided support during the project. Thanks to the people at the reasoning and learning lab at Linköping University, especially those who were active in the lab during the pre-COVID-19 phase of the project when I spent time at the university, Daniel, David, Fredrik, and Mattias, for interesting discussions about all things related to artificial intelligence. Thanks to the people at the simulation centre at Saab for help with setting up a machine learning enabled simulator, and thanks to all pilots that participated in discussions related to user needs in simulation-based training.

I completed this dissertation as an affiliated PhD student in the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP). This provided opportunities for valuable discussions with other PhD students as part of courses, conferences, and study trips. Thanks especially to the other WASP PhD students at Linköping University: Hector, Joakim, Caroline, John, Emil, Kristin, and others.

Two highlights in this project were the big international collaborations that resulted in the papers *A practical guide to multi-objective reinforcement learning and planning* and *Scalar reward is not enough: A response to Silver, Singh, Precup and Sutton (2021)*. These collaborations provided great opportunities to learn more about multi-objective reinforcement learning, and I am grateful to all the co-authors for the experience.

Johan Källström

Linköping, October 2023
## Contents

Abstract iii  
Acknowledgments vii  
Contents ix  
List of Figures xi  
List of Tables xvii  

### 1 Introduction 1

1.1 Motivation 1  
1.2 Aim 4  
1.3 Research Questions 4  
1.4 Methodology 4  
1.5 Delimitations 5  
1.6 Contributions 6  
1.7 List of Publications 6  
1.8 List of Complementary Publications 8  
1.9 Dissertation Outline 9  

### 2 Background 11

2.1 Aircraft Systems Affecting Pilots’ Decision-Making 11  
2.2 Adaptive Training 13  
2.3 Live, Virtual, and Constructive Simulation 14  
2.4 Reinforcement Learning 17  
2.5 Multi-Agent Reinforcement Learning 21  
2.6 Multi-Objective Reinforcement Learning 22  
2.7 Multi-Objective Multi-Agent Reinforcement Learning 31  
2.8 Improving the Efficiency of Learning 32
<table>
<thead>
<tr>
<th>I</th>
<th>Domain Analysis and Concept Design</th>
<th>37</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>User Study</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>Constraints Affecting Instructors</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Human-Machine Interaction in Air Combat</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>User Needs in Simulation-Based Training using Agents</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>53</td>
</tr>
<tr>
<td>4</td>
<td>Learning to Interact and Coordinate</td>
<td>55</td>
</tr>
<tr>
<td>4.1</td>
<td>The Need for Coordination in Air Combat Simulation</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>Method</td>
<td>57</td>
</tr>
<tr>
<td>4.3</td>
<td>Experimental Evaluation</td>
<td>60</td>
</tr>
<tr>
<td>4.4</td>
<td>Related Work</td>
<td>75</td>
</tr>
<tr>
<td>4.5</td>
<td>Summary</td>
<td>76</td>
</tr>
<tr>
<td>5</td>
<td>Concept Design</td>
<td>79</td>
</tr>
<tr>
<td>5.1</td>
<td>Architecture for Training Systems using Learning Agents</td>
<td>79</td>
</tr>
<tr>
<td>5.2</td>
<td>Multi-Objective Learning for Training Simulations</td>
<td>89</td>
</tr>
<tr>
<td>5.3</td>
<td>Summary</td>
<td>92</td>
</tr>
<tr>
<td>II</td>
<td>Utility-Based Reinforcement Learning for Simulation</td>
<td>95</td>
</tr>
<tr>
<td>6</td>
<td>Tunable Dynamics in Agent-Based Simulation</td>
<td>97</td>
</tr>
<tr>
<td>6.1</td>
<td>Providing Decision Support to Instructors</td>
<td>97</td>
</tr>
<tr>
<td>6.2</td>
<td>Related Work</td>
<td>99</td>
</tr>
<tr>
<td>6.3</td>
<td>Method</td>
<td>100</td>
</tr>
<tr>
<td>6.4</td>
<td>Experimental Evaluation</td>
<td>102</td>
</tr>
<tr>
<td>6.5</td>
<td>Summary</td>
<td>122</td>
</tr>
<tr>
<td>7</td>
<td>Dreaming with Dynamic Utility Functions</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>Adapting Online to Changes in User Utility</td>
<td>125</td>
</tr>
<tr>
<td>7.2</td>
<td>Related Work</td>
<td>127</td>
</tr>
<tr>
<td>7.3</td>
<td>Method</td>
<td>129</td>
</tr>
<tr>
<td>7.4</td>
<td>Experimental Evaluation</td>
<td>134</td>
</tr>
<tr>
<td>7.5</td>
<td>Summary</td>
<td>143</td>
</tr>
<tr>
<td>8</td>
<td>Conclusion</td>
<td>145</td>
</tr>
<tr>
<td>8.1</td>
<td>Summary of Contributions</td>
<td>145</td>
</tr>
<tr>
<td>8.2</td>
<td>Expected Impact</td>
<td>147</td>
</tr>
<tr>
<td>8.3</td>
<td>Future Work</td>
<td>148</td>
</tr>
<tr>
<td>8.4</td>
<td>Final Words</td>
<td>150</td>
</tr>
<tr>
<td>Bibliography</td>
<td>151</td>
<td></td>
</tr>
</tbody>
</table>
## List of Figures

1.1 An LVC distributed simulation, integrating Live aircraft, Virtual flight simulators, and Constructive simulations that provide synthetic actors controlled by artificial intelligence. .................................................. 2

1.2 Users of simulation-based training systems. .................................................. 3

1.3 Three interacting iterative processes identify user needs, develop concepts that could meet those needs, and then evaluate concepts in experiments. .................................................. 5

2.1 Fighter aircraft cockpit (Photo: Saab AB, copyright Saab AB). ......................... 13

2.2 Process for simulation-based training. Content for training sessions is created based on input regarding, e.g., training goals and trainees’ proficiency. Then, when conducting training, trainees are briefed about content and goals of the session, the session is executed, and the outcome is discussed in a debriefing. .................................................. 14

2.3 Virtual flight simulator cockpit with dome projection system showing the aircraft’s surroundings (Photo: Per Kustvik, copyright Saab AB). .................................................. 16

2.4 Markov decision process. The agent executes an action $a_t$ based on the current state $s_t$ and its decision-making policy $\pi$. The transition dynamics of the environment determine which state $s_{t+1}$ the agent ends up in, and the environment’s reward function provides a reward $r_t$ as feedback for learning an improved policy. .................................................. 18

2.5 Solution sets for multi-objective reinforcement learning [45]. ......................... 26

2.6 Example of a hyper-volume calculation, with a reference vector in blue, dominated solutions in black, and undominated solutions in red. The green solutions illustrate the effect on the hyper-volume by adding two new undominated solutions to different parts of the front [45]. .................................................. 27

2.7 Workflow for a utility-based approach to multi-objective reinforcement learning and planning [45]. .................................................. 28

2.8 Potential-based reward shaping for a navigation task, where the agent is rewarded for moving to states with higher potential. .................................................. 34
2.9 **Left:** Example of curriculum learning with increasing difficulty in the scenario, as a result of successively adding more hostile air defence systems to the scenario. **Right:** Example of curriculum learning with changes in start distribution, where the distribution of the agent’s start states is successively moved further from the target.

3.1 Subjective ranking of constraints affecting training for different types of simulation resources

3.2 Hostile entities approaching a Combat Air Patrol (CAP) along the red route. The aircraft of the CAP are protecting the Fighter Area Of Responsibility (FAOR) in blue, where high-value assets in yellow are located.

3.3 Air combat training score for blue (in solid blue circle at top of figure) and red (in dashed red circle at top of figure) forces in a counterair operations scenario.

3.4 Importance of different types of agent behaviour

3.5 Importance of different types of agent roles and voice interaction

3.6 Importance of different types of scenario characteristics

4.1 Coordinate system used in simulation scenarios

4.2 Components of a GYM-CGF environment

4.3 Simulation scenario for Offensive Counter Air

4.4 Mean and standard deviation for the training progress of Offensive Counter Air with low-level and high-level action spaces

4.5 Simulation scenario for Defensive Counter Air

4.6 Mean and standard deviation for the training progress of Defensive Counter Air with low-level and high-level action spaces

4.7 Simulation scenario for Defensive Counter Air with Human-Agent Teaming

4.8 Illustration of the monitor setup that provides observations for human pilots in experiments with human-agent teaming. **Left:** Out-the-window view. **Right:** HDD map with view of the scenario.

4.9 **Left:** Normalised score for agent teams and mixed teams in Defensive Counter Air with Human-Agent Teaming. **Right:** Bank angle distribution for aircraft operated by agents and humans in Defensive Counter Air with Human-Agent Teaming.

4.10 Simulation scenario for Aerial Reconnaissance

4.11 Mean and standard deviation for the training progress of Aerial Reconnaissance, for open-loop curriculum learning agents with shared and mixed rewards (top row), closed-loop curriculum learning agents with shared and mixed rewards (middle row), and agents learning on fixed size search areas with shared and mixed rewards (bottom row).
4.12 Performance metrics on the goal task (search area with 20 km radius) of Aerial Reconnaissance for agents learning with curricula 1-4 and agents learning on search areas with fixed radius of 5 km, 10 km, or 20 km. **Left:** Success rate of agents learning with shared and mixed rewards, i.e. ratio of episodes where all POIs are visited. **Right:** Average number of POI visits per episode for agents learning with shared and mixed rewards.

5.1 System architecture for training system using learning agents.

5.2 Motivating scenarios for a multi-objective approach to reinforcement learning [45]: (a) the unknown utility function scenario, (b) the decision support scenario, (c) the known utility function scenario, (d) the interactive decision support scenario, (e) the dynamic utility function scenario, and (f) the review and adjust scenario.

6.1 **Left:** Architecture of the tunable actor. **Right:** Utility conditioned neural network.

6.2 **Item Gathering** environment with tunable actor in blue, agent with hard-coded preferences in pink, and three categories of items in green, red and yellow.

6.3 **Left:** Mean and standard deviation for the training progress in the **Item Gathering** environment. **Right:** Utility achieved for different agents in different simulation scenarios in the **Item Gathering** environment.

6.4 Average number of different categories of items collected in different configurations of the **Item Gathering** environment. From top row to bottom row: green items, red items, yellow items, and other agent’s items. From left column to right column: Scenario 1 to scenario 4. Brighter colours indicate higher values. **Left:** Metrics for tunable actor. **Right:** Metrics for agents with fixed objective preferences.

6.5 **Traffic Navigation** environment with tunable actor in blue, items to collect in green, scripted agents in red, walls in white and road in yellow.

6.6 **Left:** Mean and standard deviation for the training progress in the **Traffic Navigation** environment. **Right:** Utility achieved for different agents in different simulation scenarios in the **Traffic Navigation** environment.
6.7 Average number of different categories of events occurring in different configurations of the Traffic Navigation environment. From top row to bottom row: steps taken, objects collected, traffic violations committed, and collisions with cars. From left column to right column: Scenario 1 to scenario 4. Brighter colours indicate higher values. **Left:** Metrics for tunable actor. **Right:** Metrics for agents with fixed objective preferences.

6.8 Routes in the Traffic Navigation environment for tunable actor and agents with fixed objective preferences. Brighter areas have been visited more often.

6.9 Simulation scenario for Risk-Aware Flight Route Selection.

6.10 **Left:** Training progress for tunable actor. **Right:** Training progress for a set of fixed agents with different priorities among objectives.

6.11 Average accumulated rewards for individual objectives in the Risk-Aware Flight Route Selection environment for tunable actor and agents with fixed objective preferences. Brighter areas correspond to higher rewards. **Left:** Performance of the tunable actor. **Right:** Performance of agents with fixed objective preferences.

6.12 **Left:** Mean and standard deviation for accumulated utility for tunable actor and fixed agents. **Right:** PCS and CCS learned by tunable actors over the iterations of the experiment.

6.13 Learned routes to the target area with high, medium and low risk exposure, for tunable actor (t) and fixed policies (f).

6.14 Simulation scenario for Air Interdiction with Fighter Escort.

6.15 **Left:** Training progress for tunable actors learning with MADDPG and DDPG. **Right:** Training progress for agents with fixed utility functions learning with MADDPG.

6.16 Average accumulated rewards for individual objectives in Risk-Aware Air Interdiction with Fighter Escort. Brighter areas have higher average rewards.

6.17 **Left:** Achieved average utility for tunable actors learning with MADDPG and DDPG, and MADDPG agents learning with fixed preferences over objectives. **Right:** Emergent behaviour for safe (green trajectories) and risk-taking (red trajectories) tunable actors learning with MADDPG.

7.1 MO-Dreamer interacts with the environment and builds a dataset of diverse experiences, which are used to construct a model for imagination rollouts where past experienced states are revisited with experienced as well as imagined utility functions for improvement of the policy.
7.2 From left to right, Minecart, Partially Observable Minecart, Deep Sea Treasure, and Partially Observable Deep Sea Treasure evaluation environments ........................................ 134

7.3 Average cumulative episodic regret on Minecart over ten runs for frequent (left) and sparse (right) utility changes .................. 138

7.4 Average cumulative episodic regret on Deep Sea Treasure over ten runs for frequent (left) and sparse (right) utility changes, and full (top row) as well as partial observability (bottom row) .......... 140

7.5 Agent observations from the steps of an episode in the Minecart environment .................................................................. 142
List of Tables

4.1 Fixed, open-loop curricula used in the Aerial Reconnaissance scenario. The Switches lists specify at which time steps each Search Radius Interval \((r_0-r_3)\) is activated within a curriculum. The radius intervals are given in kilometres. .......................................................... 71

7.1 Average episodic regret on Minecart for sparse and frequent utility changes. ................................................................. 139

7.2 Average episodic regret on Deep Sea Treasure for sparse and frequent utility changes. ................................................................. 141
1 Introduction

Team training in complex domains often requires a substantial amount of resources, e.g. vehicles, role-players, and instructors. For this reason, it may be difficult to realise efficient and effective training scenarios in a real-world setting. Instead, part of the training can be conducted in synthetic, computer generated environments. In these environments trainees can operate simulators instead of real vehicles, while synthetic actors can replace human role-players to increase the complexity of the simulated scenario at low operating cost. However, constructing agents that can control such actors is challenging \cite{136,14,37}, especially for the end-users of the training systems, who typically do not have expertise in artificial intelligence. In this dissertation, we study how machine learning can be used to simplify the construction of intelligent agents, whose behaviour can be adapted to the needs of human trainees. By constructing smarter agents the dependency on human training providers can be reduced, while the availability and quality of training is improved. As a case study, we use a simulation-based air combat training system, which presents many challenges to synthetic agents in terms of completing tasks in the simulated world as well providing efficient and effective training to the human trainees.

1.1 Motivation

Providing efficient and effective training solutions for fighter pilots is becoming increasingly challenging. Due to the high operational cost of aircraft, limited
availability of air space, and strict safety regulations, it is difficult to realise training scenarios with the desired contents and density in a live setting. In addition, conducting training in the open results in a risk of revealing sensitive information about tactics and systems performance. To address these challenges, virtual and constructive simulation resources must be used to a higher degree. Live, Virtual and Constructive (LVC) simulation aims to integrate real aircraft, ground-based systems and soldiers (Live), manned simulators (Virtual), and computer-controlled synthetic entities (Constructive) \[10\], as illustrated in Figure 1.1. By using constructive simulation to augment the live and virtual aircraft operated by trainees and role-players, it is possible to improve training efficiency and effectiveness by simulating scenarios with a large number of participating friendly, hostile and neutral entities \[51, 111, 55\]. Ideally, synthetic agents should be able to act as trainers, and adapt their behaviour to the training needs of the human trainees. This would allow us to minimise the number of human support personnel required for conducting training, which would lead to lower costs and improved availability of training.

As illustrated in Figure 1.2, we can divide the users of training systems into two major categories: training audience and training providers. The training audience consists of those in training, e.g. pilots learning how to operate a new aircraft. The training providers consist of those delivering the training, such as instructors, role-players, scenario operators and scenario developers. Instructors are responsible for the pedagogical contents of a training session, while role-players and scenario operators help deliver the training by participating as actors or by controlling parts of the simulated scenario respectively. The scenario developer is responsible for developing the scenarios that should be simulated, including behaviour models for controlling synthetic actors. In practice, one single person could act in several roles. For instance, due to limited resources, one person could act as instructor, role-player and operator.
1.1. Motivation

This typically results in a high workload, and the desired training scenarios may not be achievable. It is desirable to reduce the need for training providers, to improve training efficiency as well as effectiveness. If synthetic agents were to become smarter, they could replace or augment human role-players, and reduce the amount of input required from scenario operators for the training scenario to progress in the desired way. To further raise the level of autonomy of the system, synthetic agents could also assist instructors in evaluating the performance of the trainees, and in adapting the contents and characteristics of training scenarios [16]. However, creating behaviour models for the agents is challenging, especially for end-users of training systems, e.g. instructors, who may not have the required expertise and experience [37]. In the past, this has constrained the use of agents in training. Now, with the recent advances in artificial intelligence (AI), there is hope that data driven methods will simplify the process of constructing intelligent agents, which could replace or assist human support personnel in simulation-based training.

For learning sequential decision-making, reinforcement learning [128] has become the state of the art method. Guided by a human-designed reward signal, such agents can learn a decision-making policy purely by interacting with their environment. By leveraging deep learning [38] in combination with reinforcement learning, it has become possible to beat human champions in classic board games as well as multi-player computer games [122, 123, 61, 148]. The results have sparked interest in investigating applications of reinforcement learning in many domains, including air combat simulation, where learning agents could act as teammates as well as opponents. While reinforcement learning research in general often focuses on achieving superhuman performance in some domain, e.g. games, this work does not focus on using reinforcement learning for development of new air combat tactics, but instead studies how agents can learn behaviour that improves the performance of the humans that participate in simulation scenarios. In our development of new agents, we focus on multi-objective reinforcement learning, which can find trade-offs between multiple conflicting objectives, and in particular the utility-based approach, which uses knowledge about the users' utility functions to make learning more efficient and effective.
1. Introduction

1.2 Aim

The aim of this dissertation is to analyse and model simulation-based training systems incorporating synthetic learning agents for improvement of efficiency and effectiveness; and to support delivery of adaptive simulation-based training by developing methods for utility-based reinforcement learning, which aims to provide an optimal solution based on knowledge about the application domain and the utility functions of its users. The long-term goal is to enable efficient adaptation of training to the needs of each individual trainee, instead of providing a one-size-fits-all solution.

1.3 Research Questions

To achieve the aim of the dissertation, we address the following research questions:

RQ1: What capabilities and characteristics do synthetic agents need to participate as actors in mixed cooperative and competitive simulation-based training scenarios?

RQ2: How can reinforcement learning methods assist an instructor in delivering training?

RQ3: How can simulation contents be adapted to fit the training needs of an individual trainee?

Research question RQ1 focuses on reinforcement learning agents acting as synthetic role-players, and interacting with the human trainees to efficiently provide an effective training environment. Research question RQ2 focuses on reinforcement learning agents acting as synthetic assistants, and supporting the work of the instructor. Research question RQ3 concerns the desired characteristics of synthetic actors as well as synthetic instructor assistants when they are deployed in a user-adaptive training environment.

1.4 Methodology

The methodology used in the dissertation work consists of three iterative and interactive processes: domain analysis, concept development, and experimental evaluation, as illustrated in Figure 1.3. The domain analysis is intended to improve the understanding of the application domain including its users and their needs, to guide concept development in the most promising direction. A reference group with subject matter experts (SMEs) from industry and the air force is used to support this process. The concept development is intended to study and compare a few promising concepts in more detail, for further down-selection and specialisation before experimental evaluations are conducted. In
experimental evaluations concepts are studied in simple simulation scenarios to find their strengths and weaknesses. The output from concept development and experimental evaluation are used to support further domain analysis in a feedback loop.

1.5 Delimitations

The methods that we study in this dissertation are evaluated in simulation scenarios that are simple compared to those that would be used in actual training. The reason for this is that current reinforcement learning algorithms require substantial computation resources for agents to find decision-making policies in complex environments. In addition, most evaluations do not include human participants, because of the large number of participants required to produce statistically significant results. The experiments should thus be viewed as an initial evaluation of ideas and concepts. By using simpler scenarios, more iterations can be completed according to the methodology presented in Section 1.4. More advanced, validating experiments should ideally be conducted in combination with other research efforts and as part of operational training, once the studied methods have been developed to a higher level of maturity.
1. Introduction

1.6 Contributions

The contributions of this dissertation can be summarised as follows:

C1: A domain analysis was conducted to identify user needs and desirable agent capabilities in different types of air combat training scenarios. The analysis consisted of user interviews, a written survey, and practical experiments in the target system. This analysis helps address research questions RQ1 and RQ2.

C2: A system concept was developed based on the results from the domain analysis, to frame future research efforts in the domain. The system concept proposes an architecture for a user adaptive training system, which aims to continuously adjust the simulated environment to fit the training needs of individual trainees. The system concept helps address research questions RQ1, RQ2, and RQ3.

C3: A Tunable Actor (T-Actor) agent was developed. T-Actor provides decision support to instructors by learning a set of policies for a parameterised utility function. The utility function models the agent’s preferences over a set of objectives, and affects its behaviour. An instructor can select a suitable policy for an individual trainee by a posteriori tuning of the utility function. T-Actor helps address research questions RQ2 and RQ3.

C4: A multi-objective model-based actor-critic, Multi-Objective Dreamer (MO-Dreamer), was developed. MO-Dreamer learns online in environments with dynamic utility functions, and uses imagination rollouts in a learned world model to improve its policy in anticipation of utility changes. MO-Dreamer helps address research questions RQ2 and RQ3.

1.7 List of Publications

The contributions of this dissertation are to a large extent based on the material presented in the following list of publications. For papers where nothing else is stated, I developed the idea, conducted the experiments, and wrote the manuscript. Fredrik Heintz provided feedback on the manuscripts.


1.7. List of Publications


**Individual contributions**: As the main author of this paper, I developed the idea, conducted the experiments, and wrote the manuscript. Rego Granlund participated in the design of a user study, which consisted of a survey and an experimental study of human-agent interaction.


**Individual contributions**: This paper was the result of a large collaboration, which I was invited to participate in. The paper provides a guide to multi-objective reinforcement learning, with a focus on the utility-based approach. I primarily contributed to the sections on methods for high-dimensional observations, interactive methods, and benchmarks for evaluation. In addition, I provided feedback regarding the content and structure of other sections, and helped address reviewers' comments.

1. Introduction


1.8 List of Complementary Publications

In addition to the publications that form the foundation of this dissertation, the following publications were also produced, but their contents will not be discussed in detail in the dissertation.


**Individual contributions:** This paper was the result of a project course, where design of decision support systems for search and rescue operations was studied. I planned and conducted a field study, which investigated users’ current situation, and collected feedback on a number of proposed design improvements. In addition, I developed a scenario visualisation tool based on augmented reality, and together with Fredrik Präntare I developed another scenario visualisation tool for standard monitors.


**Individual contributions:** This paper was proposed by Peter Vamplew. I primarily contributed to discussions related to social intelligence and decision-making in multi-agent systems. In addition, I provided feedback regarding the content and structure of other sections, and helped address reviewers’ comments.

**Paper X** discusses design of decision support systems intended for search and rescue operations where autonomous agents, e.g. drones, are used. Such support systems have many requirements and features in common with the instructor/operator stations used in facilities for simulation-based air combat training. The design of the user interface is important to enable an instructor/operator to control the flow of the tactical scenario during a training session, e.g. by sending commands to semi-autonomous entities in the scenario. However, user interface design is not the focus of this dissertation. **Paper XI**
complements Paper VII by providing an extended discussion on the benefits of a multi-objective approach to reinforcement learning, e.g. in relation to ethical aspects of future development of artificial general intelligence.

1.9 Dissertation Outline

The remainder of this dissertation is organised as follows. First, Chapter 2 provides relevant background information about the application domain and reinforcement learning. Then, the contributions of the dissertation are presented in two parts.

Part I of the dissertation aims at improving the understanding of reinforcement learning in the domain of simulation-based training, in particular simulation-based training of fighter pilots. First, Chapter 3 presents a user study, which was conducted to identify important capabilities and characteristics of learning agents that are intended to support training of fighter pilots. The results of the study can be viewed as a high-level requirements specification for synthetic, learning actors for use in simulation-based air combat training. Second, Chapter 4 studies methods for learning how to coordinate with other agents in simplified training scenarios. Coordination is important in all air combat simulation, since pilots never fly alone. The performance of reinforcement learning algorithms in the studied scenarios provide insight regarding which methods are suitable for meeting the user needs identified in Chapter 3. Third, based on the results from the user study and the practical experiments, Chapter 5 develops a system concept for a user-adaptive training system, to support further research. Methods for meeting user needs are discussed in the context of the system’s architecture.

Part II of the dissertation focuses on methods for utility-based multi-objective reinforcement learning, which incorporates knowledge of the user’s utility function in the search for policies that balance multiple conflicting objectives. Two new agents are proposed based on the system concept from part I. First, Chapter 6 proposes a Tunable Actor (T-Actor) for agent-based simulation. T-Actor provides decision support to instructors by learning a set of Pareto optimal policies, represented by a single neural network conditioned on objective preferences. This enables a post-training tuning of the agent’s behaviour to fit individual trainees’ current training needs. Second, Chapter 7 presents the Multi-Objective Dreamer (MO-Dreamer). MO-Dreamer adapts online to changes in users’ utility, e.g. changes in training needs, by learning a model of the environment, which it can use for anticipatory simulations with a diverse set of utility functions. This allows the agent to prepare for future interactions with users in a wide range of different simulation scenarios.

Finally, Chapter 8 concludes the dissertation with a summary and suggested directions for future work.
2 Background

This chapter provides background information for the research conducted in this dissertation. First, information related to air combat training is presented in Sections 2.1 to 2.3, including aircraft systems, adaptive training, and simulation approaches. Then, information related to reinforcement learning is presented in Sections 2.4 to 2.8, including the basic principles, methods for learning in multi-agent systems, methods for learning with multiple objectives, and methods that can improve the efficiency of learning. This chapter is based on Paper II [71], Paper III [72], Paper VI [67], and Paper VII [45].

2.1 Aircraft Systems Affecting Pilots’ Decision-Making

In air combat scenarios, teams of pilots compete to establish control over contested air space. In this section we briefly describe aircraft systems that are used by pilots to establish situational awareness and execute mission plans. A synthetic, learning agent must be able to learn how to operate these systems to be competitive in air combat simulations.

Radar and electro-optical Sensor systems enable detection and positioning of vehicles in the surroundings of the aircraft. Active sensors, which emit energy, can provide better and more reliable performance than passive sensors. However, emitting energy makes it easier for an enemy to detect your aircraft. This results in a trade-off between searching for others and keeping yourself hidden.
2. Background

Communication systems allow pilots and operators of other units to share information and coordinate execution of mission plans. Communication can be divided into radio voice communication, where participants can communicate freely, and data link communication, which is governed by communication protocols. As for sensor operations, communication results in emission of energy, which can help enemies detect the aircraft.

Weapon systems allow pilots to attack targets in the air or on the ground. Some weapons have better performance than others (e.g., longer range or higher precision), but that performance boost typically results in higher cost. Weapon delivery procedures also vary between weapons. For instance, some weapons have a “fire and forget” delivery mechanism, which allows the pilot to turn away from threats directly after firing. Other weapons require that the pilot guides the weapon towards the target for some time after firing, until the weapon is close enough to the target for the pilot to handover control to the weapon (determined by the range of the weapon’s target seeker). This results in an increased risk of being detected and attacked by enemies if the weapon was fired in a high-risk area. Pilots must use their situational awareness to decide when to use which weapon to balance, e.g., weapon effect, cost, and safety.

Electronic Warfare systems make it possible to detect and identify other systems based on the signatures of their electromagnetic emissions. Knowing the type of a detected entity, e.g. aircraft type, helps the pilot make an informed decision about how to deal with that entity. If a threat is detected, electronic countermeasures can be used to reduce the performance of the threat’s sensors by attacking them with noise jamming or deception jamming (e.g. transmitting pulses that appear as false target at the receiving end). Noise jamming can also degrade the performance of enemies’ communication systems. Evasion of incoming missiles can be supported by dispensing chaff against radar homing missiles or flares against infrared homing missiles.

Control and Display systems allow pilots to control the aircraft and observe information fused from the aircraft sensors. Figure 2.1 illustrates the cockpit of a fighter aircraft. Controls and displays are organised so that those used more frequently are more easily accessible (typically closer to the front of the cockpit). Head Down Displays (HDDs) are mounted below the lower edge of the canopy, and provide information about known entities, as well as system status. The Head Up Display (HUD) mounted in the centre of the canopy provides important information to the pilot while looking forward out the cockpit, e.g. symbology to support navigation or weapon aiming, while a Helmet Mounted Display (HMD) makes it possible to provide symbology based on the position and orientation of the pilot’s head. Rear-view mirrors make it possible for the pilot to observe what happens behind the aircraft, e.g., when engaged in close-range combat.

When the complexity of aircraft systems and the operational environment increase, the need for efficient and effective training environments also in-
creases. In addition, with increasing complexity it becomes increasingly less likely that a one-size-fits-all training approach will be sufficient for all pilots to reach the desired level of proficiency. For instance, depending on the background of each individual trainee, some may require more training in precise execution of flight manoeuvres, while others may benefit more from training sessions that focus on operation of tactical systems. Having greater flexibility in adapting training contents to the needs of each individual can be expected to improve the utility of training.

2.2 Adaptive Training

In adaptive training, the training scenarios are continuously adapted based on measurements of trainees’ performance \[\text{(74)}\]. When the performance of trainees improves, the training task is made more challenging. If the performance of trainees is poor, the challenge of the training task is reduced. In contrast, in a fixed training system, a typical trainee might feel that the training task is too challenging in the initial stages of training, suitably challenging in a short middle stage of training, and too easy in the final stages of training. By adapting the training task to each individual trainee, trainees can improve their performance faster.

The adaptation of training can be handled by a human instructor. A typical training process for a simulation-based training system is illustrated in Figure 2.2. First, simulation contents are created to meet identified training needs. In the domain of air combat this could include vehicle models, behaviour models for the synthetic operators of these vehicles, and definitions of the scenarios that they operate in. Then, in a training session, a briefing
2. Background

Content Creation
Training Session
Briefing
Execution
Debriefing
External Input Feedback

Figure 2.2: Process for simulation-based training. Content for training sessions is created based on input regarding, e.g., training goals and trainees’ proficiency. Then, when conducting training, trainees are briefed about content and goals of the session, the session is executed, and the outcome is discussed in a debriefing.

is conducted to present and discuss training objectives and scenario contents, followed by the actual execution of the scenario. Afterwards, trainee performance is evaluated in a debriefing to allow for reflection among the participants. Over time, training needs are updated based on the learning progress of trainees, as well as input from the organisation that they belong to, e.g. due to changes in operational missions.

Efficiency can be improved by automating the adaptation process in an adaptive training system. An adaptive training system has three major elements: (1) a way to measure the performance of trainees, (2) an adaptive variable that can be adjusted to change the difficulty of the training scenario, and (3) adaptive logic that affects the adaptive variable based on the performance measurements [74]. In team training, all participating trainees must be considered by the performance measurements and the adaptive logic. The level of challenge related to adapting the training environment depends on the types of simulation resources used, as discussed in the next section.

2.3 Live, Virtual, and Constructive Simulation

Operation of fighter aircraft is highly expensive. Based on open data, the cost per flying hour (CPFH) for modern fighter aircraft can be estimated to range from US$4,000 to as much as US$40,000, depending on the type of aircraft (e.g. single-engine or twin-engine aircraft) and the types of costs included in the estimate (e.g. costs for a single flight, or average costs for the lifespan of the aircraft) [59, 64, 57]. Computer simulations can be used to complement training in live systems, to reduce costs. One approach is to replace some of the live training with training in simulators, e.g. using ground-based flight simulators instead of training in live aircraft. Training in simulators before training in the air can also improve the quality of live training, by ensuring that pilots are prepared for the training task [6]. As an example of the cost of
training in a simulator, the cost of training of airline pilots can be estimated
to range from US$500 to US$1000 per hour, all expenses included, while the
cost of buying the simulator itself could be around US$10 million [79]. Ac-
cording to an estimate from the Swedish Air Force Combat Simulation Centre
(Flygvapnets Luftstrids Simulerings Centrum, FLSC), the Swedish air force
can save roughly 6000 flying hours per year by using a simulation facility with
support for many-to-many air combat training [83]. Another approach is to
embed simulation capabilities in live systems, e.g. capabilities for generation
of synthetic opponents in a fighter aircraft, so that fewer aircraft are required
to conduct training. It has been estimated that embedded training can im-
prove training effectiveness of live training by 30% at the same cost [111]. In
the Live, Virtual, and Constructive simulation paradigm, the goal is to take
things one step further, by seamlessly integrating live systems, manned sim-
ulators and computerised simulations in a distributed simulation. The three
categories of simulations are defined as [100]:

- **Live**: Simulations involving real people operating real systems
- **Virtual**: Simulations involving real people operating simulated systems
- **Constructive**: Simulations involving simulated people operating simu-
  lated systems (possibly stimulated by real people)

In the domain of air combat simulation the goal is to integrate real air-
craft, ground-based systems and soldiers (Live), manned simulators (Virtual)
and computer controlled entities (Constructive), which can improve training
value [51, 55]. As noted above, Live simulation in the air domain is highly
expensive. In addition, it requires access to air space, needs to adhere to strict
safety regulations, and may reveal sensitive information to an observer. Costs
can be reduced by replacing some of the aircraft with virtual flight simula-
tors. These can still have the correct controls and displays in the cockpit,
although they interact with a simulated world. Figure 2.3 shows an example
of a virtual flight simulator with a dome projection system.

Constructive simulation enables realisation of complex scenarios with a
large number of autonomous friendly, hostile and neutral entities, which inter-
act with each other as well as manned simulators and real systems. However,
building realistic behaviour models for constructive simulation is a significant
challenge [136, 14, 137], and consequently support from scenario operators,
and possibly human role-players, is still required in many training scenar-
ios. Consequently, users may decide to allocate simpler, pre-defined tasks to
constructive entities, or simply use them as background noise [2, 106, 159].

With improved behaviour models, constructive entities could more easily act
in central roles in training scenarios, training systems with a higher level of
autonomy could be built, and adaptive training with contents tailored to the
current learning needs of individual trainees could be provided [22, 10].
2. Background

Due to the difficulties associated with manual construction of behaviour models for constructive entities, there has been interest in using various forms of machine learning for constructing these models, e.g., dynamic scripting \[140, 139, 138\], evolutionary algorithms \[21, 157, 165\], and neural networks \[134, 135\]. However, the techniques studied have not been mature enough for inclusion in commercial software for military simulation \[2, 137\].

With the advancement of artificial intelligence in recent years, there is renewed and rapidly increasing interest in machine learning, in particular for the branch of machine learning known as reinforcement learning. It has been shown that reinforcement learning can be used to create synthetic agents with human-level performance in games that share some characteristics with air combat scenarios \[61, 99, 148\], indicating that this technique could be valuable for development of synthetic fighter pilots as well.

The domain of air combat training presents many challenges to learning agents. Air combat is a mixed cooperative and competitive problem, where teams of agents compete. In training scenarios, synthetic agents must have the ability to interact with humans as well as other synthetic agents. Limitations in sensors and data links, as well as effects of electronic warfare, result in a partially observable environment, and what can be seen is also affected by the actions of the agents. While acting in this environment, agents must consider multiple conflicting objectives, such as mission goals, resource consumption, and safety.

The following sections discuss reinforcement learning methods that are of relevance for learning in the air combat domain. These methods are then...
further discussed, evaluated, and extended in the remaining chapters of the dissertation.

2.4 Reinforcement Learning

Reinforcement learning (RL) allows an agent to learn a function for decision-making (policy $\pi$) by interacting with its environment in a form of trial-and-error learning [128]. A reinforcement learning problem is often modelled as a Markov Decision Process (MDP) [128], or variations thereof.

Definition 1 An MDP is defined by a tuple $(S, A, T, R, \gamma, \mu)$, where:

- $S$ is the state space
- $A$ is the set of actions
- $T: S \times A \times S \rightarrow [0, 1]$ is a probabilistic transition function
- $R: S \times A \times S \rightarrow \mathbb{R}$ is the reward function
- $\gamma \in (0, 1)$ is a discount factor indicating the importance of immediate and future rewards respectively
- $\mu: S \rightarrow [0, 1]$ is a probability distribution over initial states

The agent interacts with its environment by selecting actions according to its policy ($a_t = \pi(s_t)$), and observes the resulting environment state ($s_{t+1}$) and the reward received based on its action ($r_t = R(s_t, a_t, s_{t+1})$). When the agent executes an action that results in high reward, that action is reinforced, so that it will be taken more often in the future. During learning, the agent must balance between exploration and exploitation, which is one of the greatest challenges of reinforcement learning. Exploration means that the agent selects exploratory actions to learn more about the environment, while exploitation means that the agent uses the knowledge gained so far to gather reward. Learning can be on-policy or off-policy. For on-policy methods, the policy used for exploration (the behaviour policy), is the same as the policy being optimised (the target policy). For off-policy methods, the behaviour policy can be different from the target policy, e.g., the behaviour policy may sometimes take random actions to promote exploration. The process of reinforcement learning is illustrated in Figure 2.4.

The goal of the agent is to maximise its future expected return $G_t$ when starting in state $s_t$ and then following policy $\pi$, which is captured in the state value function $V^\pi(s)$:

$$V^\pi(s) = \mathbb{E}[G_t|s_t = s] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s\right]$$ (2.1)
2. Background

We can also define a state-action value function $Q$, which specifies the value of taking action $a$ in state $s$ and then following policy $\pi$:

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

(2.2)

The $Q$ function can be used as a policy, by greedily selecting the action $a_t$ with highest estimated value for the current state $s_t$. Methods that learn a value function are called value-based. In contrast, methods that learn a policy without learning a value function are called policy-based. The $Q$ function can be learned through Q-learning [154], by representing the $Q$ function as a table of $Q$ values, and applying the following update rule (with learning rate $\alpha$) in each step of the episode:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha (r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

(2.3)

The tabular approach to reinforcement learning does not scale well to complex state and action spaces, which limits its applicability to many real-world problems. Instead, function approximation can be used. The performance of learning agents has been improved rapidly by combining reinforcement learning with function approximation based on deep learning. The goal of deep learning is to make it possible to use data to create models that describe the contents of their input with several connected layers of successively more abstract concepts [38]. There are different types of neural network architectures, which are more or less suitable for handling certain types of input data. In reinforcement learning, commonly used network types are Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs).

Multi-layer perceptrons, also known as feedforward networks, can be trained to approximate a function $y = f(x; \theta)$ by optimising the parameter vector $\theta$ to minimise the approximation error for a dataset with exam-
Feedforward networks contain a sequence of layers of neurons. Each neuron in a layer has a connection to every neuron in the next layer, and data flows from the input layer to the output layer while passing a number of internal, hidden layers. To allow for modelling of non-linear functions, the mapping from input to output data is implemented as a linear model applied on a non-linear transformation of the input data $y = f(x; \theta, \omega) = \phi(x; \theta)^\top \omega$, where $\phi(x; \theta)$ is a representation of the input $x$ learned by the network. In a multi-layer network, each layer learns a representation by applying a non-linear activation function on the result of an affine transformation $h = g(W^\top x + c)$, where $W$ and $c$ are learned parameters, while the structure of the network and the choice of activation function $g$ are design decisions. A commonly used activation function is the rectified linear unit (ReLU), which is defined as $g(z) = \max\{0, z\}$.

Convolutional neural networks are a form of feedforward networks designed to operate on data with a grid-like structure, such as time-series data and images. In each layer, convolutions with sparsely connected filters are used to extract features from different parts of the input-grid. When moving from the input layer into the deeper layers of the network, successively more abstract features can be detected, e.g., first detecting edges and then detecting faces in an image. The sparse connections make training of deep CNNs more efficient than training of fully connected MLPs.

Recurrent neural networks extend feedforward networks by adding feedback connections, and make it possible to model sequences of data. For instance, in reinforcement learning, a sequence of observations can be used as input to support the decision-making of the agent, instead of requiring it to make decisions based only on the information available in a single time step. This is especially important in environments whose true state is only partially observable.

A breakthrough in reinforcement learning was the development of the Deep Q Networks (DQN) algorithm, which uses a deep convolutional neural network with parameters $\theta$ to estimate the optimal state-action value function $Q(s, a; \theta) = Q^*(s, a)$, making it possible for agents to learn how to play classic video games from pixels. DQN uses experience replay to train the neural network. In each time step in the environment, the experience tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ is stored in an experience data set $D$. Experiences are then sampled periodically to minimise the loss:

$$L_i(\theta_i) = \mathbb{E}_{e \sim D}[(y_i - Q(s, a; \theta_i))^2],$$

using stochastic gradient descent. The target value of the update at iteration $i$ is:

$$y_i = r_t + \gamma \max_a Q(s_{t+1}, a; \theta^*_t),$$
where $\theta_i^-$ are the parameters of a target network, which are periodically synchronised with the parameters $\theta$ of the $Q$ network. The experience replay mechanism and the target network help stabilise learning by reducing the effect of correlations in the sequences of interactions with the environment as well as correlations between action values and target values. As one of the early scalable deep reinforcement learning algorithms, it has been evaluated in many application domains, including air combat simulation [114, 83, 86, 169, 71]. Over the years the algorithm has been improved with several extensions [52]. For instance, prioritised experience replay prioritises sampling experiences where there is a large temporal difference (TD) error, instead of sampling uniformly from the replay buffer [117]. The TD error is the difference between the agent’s current estimate of an action value and its target value. Duelling DQN uses a network architecture with two branches in the network head [153]. One of the branches learns state values, while the other branch learns the advantage of each action. These two components are then combined to output $Q$ values. Prioritised experience replay as well as duelling DQN have been shown to provide a significant performance boost [52].

The DQN algorithm trains a network to output the values of a discrete set of actions for a given state, which may limit its applicability to, e.g., robotics problems. The Deep Deterministic Policy Gradient (DDPG) algorithm extended deep reinforcement learning to domains with continuous actions [82]. It is an actor-critic architecture, that uses a deep $Q$ critic network $Q(s, a; \theta^Q)$ to estimate the values of actions, to guide updates of the actor network $\mu(s; \theta^\mu)$. As DQN, DDPG is an off-policy algorithm that trains the agent networks with data sampled from an experience replay buffer. The target for the $Q$ function update is:

$$y_i = r_t + Q(s_{t+1}, \mu(s_{t+1}; \theta^\mu^-); \theta^Q^-),$$

The policy gradient is calculated as:

$$\nabla_{\theta^\mu} J = \frac{1}{N} \sum_i \nabla_a Q(s, a; \theta^Q) |_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s; \theta^\mu) |_{s=s_i}. (2.7)$$

The target network parameters $\theta^\mu^-$ and $\theta^Q^-$ slowly follow the parameters $\theta^\mu$ and $\theta^Q$ of the actor and critic networks when they are updated:

$$\theta^Q^- \leftarrow \tau \theta^Q + (1 - \tau) \theta^Q^-,$$

$$\theta^\mu^- \leftarrow \tau \theta^\mu + (1 - \tau) \theta^\mu^-.$$ (2.8) (2.9)

In air combat simulation, continuous actions can be valuable for platform manoeuvring, and there have been a number of studies, primarily for within visual range (WVR) combat scenarios [162, 72, 58, 81, 149].
2.5 Multi-Agent Reinforcement Learning

In air combat scenarios, pilots do not act on their own, but instead must cooperate with allies to achieve their goals while competing with enemies. To train teams of agents, multi-agent reinforcement learning can be used. The single agent MDP can be extended to include multiple agents who interact with the environment as well as each other in so called stochastic games (SG), where the environment state as well as the rewards of the individual agents are determined by the joint actions of all agents [20]. In many settings, agents can only observe parts of the environment state, since e.g. internal states or actions of other agents may be hidden. In air combat scenarios observability could be affected by e.g. performance of sensors and data links, availability of command and control systems, and effects of electronic warfare. This more general setting can be modelled as a partially observable stochastic game (POSG).

Definition 2 A POSG is defined by a tuple \( (S, A, T, R, \Omega, O, \gamma, \mu) \), where:

- \( S \) is the state space
- \( A = A_1 \times \cdots \times A_n \) represents the set of joint actions, with \( A_i \) being the action set of agent \( i \)
- \( T : S \times A \times S \to [0, 1] \) is a probabilistic transition function
- \( R = R_1 \times \cdots \times R_n \) are the reward functions, where \( R_i : S \times A \times S \to \mathbb{R} \) is the reward function of agent \( i \)
- \( \Omega = \Omega_1 \times \cdots \times \Omega_n \) represents the set of joint observations, with \( \Omega_i \) being the observation set of agent \( i \)
- \( O = O_1 \times \cdots \times O_n \) are the observation functions, where \( O_i : S \times A \times \Omega_i \to [0, 1] \) is the observation function of agent \( i \)
- \( \gamma \in [0, 1) \) is a discount factor indicating the importance of immediate and future rewards respectively
- \( \mu : S \to [0, 1] \) is a probability distribution over initial states

Stochastic games can be characterised as fully cooperative when all agents have the same reward, and fully competitive when agents have opposite rewards. Stochastic games that are neither fully cooperative nor fully competitive are called mixed games. The special case of POSG where all agents have a shared reward is commonly referred to as a Decentralised Partially Observable Markov Decision Process (Dec-POMDP) [15], which reduces to a POMDP if there is only one agent in the environment.
Multi-agent learning presents many challenges, such as coordination among agents, multi-agent credit assignment (i.e. determining a single agent’s contribution to the success of a team of agents), and non-stationary environment dynamics as a result of multiple agents learning concurrently and changing their behaviour. Interesting behaviour may still emerge if agents are trained using single-agent algorithms \[80, 131\], especially if measures are taken to ensure that agents’ policies do not change too fast \[166, 127\]. However, performance can be improved by using algorithms specifically designed for a multi-agent setting. Lowe et al. proposed the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) algorithm, an extension of DDPG to multi-agent environments \[84\], to address the challenge of multi-agent learning. MADDPG uses a centralised $Q$ function as critic to guide updates of decentralised policies. The critic of each policy has access to additional information about other policies at training time, which can help improve performance. In particular, if the critic has access to the actions taken by other agents, the environment remains stationary even when other agents are learning and updating their policies. In MADDPG, the critic of one agent conditions its output on the observations and actions of all other agents. For a set of $N$ agents, value and policy networks are parameterised by $\theta = \{\theta_1, \ldots, \theta_N\}$.

The centralised $Q$ function is trained with the following loss for agent $i$:

$$L(\theta_i) = E_{x,a,r,x'}[(Q^\mu_i(x, a_1, \ldots, a_N) - y)^2], y = r_i + \gamma Q^\mu'_i(x', a'_1, \ldots, a'_N)|_{a'_j = \mu'_j(o_j)},$$

(2.10)

where $\mu'$ indicates target networks being used for the actor and critic, $x$ is the vector of observations for all agents, and $x'$ and $a'_j$ indicate next step observations and resulting actions. The policy gradient for agent $i$ can be calculated as:

$$\nabla_{\theta_i} J(\mu_i) = E_{x,a,r,x'}[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_a Q^\mu_i(x, a_1, \ldots, a_N)|_{a_i = \mu_i(o_i)}].$$

(2.11)

The algorithm supports continuous action spaces and mixed cooperative and competitive scenarios, and is therefore suitable for applications in the air combat domain \[167, 156, 155, 170, 63, 164, 171\]. The continuous actions can be used to manoeuvre the aircraft. The support for mixed settings means that teams of friendly and hostile pilots can be trained in the same scenario, i.e. all synthetic pilots in a scenario can be learning in parallel.

### 2.6 Multi-Objective Reinforcement Learning

Multi-objective reinforcement learning (MORL) is a generalisation of standard reinforcement learning to problems where multiple conflicting objectives must be considered. Typical air combat scenarios fit this description, since
they require that the participating pilots prioritise among objectives such as targets to attack, assets to protect, safety, and resource consumption. In training scenarios, synthetic agents could also consider the learning objective of trainees, e.g. by adapting their behaviour to fit the proficiency of the trainee. A multi-objective decision-making problem can be formalised as a multi-objective Markov decision process (MOMDP) \[114, 45\].

**Definition 3** A MOMDP is defined by a tuple \((S, A, T, R, \gamma, \mu)\), with \(d \geq 2\) objectives, where:

- \(S\) is the state space
- \(A\) is the set of actions
- \(T : S \times A \times S \to [0, 1]\) is a probabilistic transition function
- \(R : S \times A \times S \to \mathbb{R}^d\) is a vector-valued reward function, specifying the immediate reward for each of the considered \(d \geq 2\) objectives
- \(\gamma \in [0, 1)\) is a discount factor indicating the importance of immediate and future rewards respectively
- \(\mu : S \to [0, 1]\) is a probability distribution over initial states

A MOMDP provides vector rewards, where each element in the vector represents the reward for one of the objectives. As in the single-objective case, the MOMDP can be extended to more complex models, such as a multi-objective partially observable Markov decision process \[96, 125, 126, 160\] and multi-objective multi-agent systems \[103\].

The vector reward of the MOMDP results in vector returns and state value functions:

\[
V^\pi(s) = \mathbb{E}[G_t|s_t = s] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|s_t = s\right].
\] (2.12)

In contrast to single-objective settings, this value function does not provide a complete ordering of policies, since there may be settings where \(V^\pi_i > V^{\pi'}_i\) for objective \(i\) while at the same time \(V^\pi_j < V^{\pi'}_j\) for objective \(j\). To determine an ordering of policies, we need information about the user’s preferences over the objectives. These preferences can be expressed in the form of a utility function \(u\), which turns the multi-objective value into a scalar:

\[
V^\pi_u(s) = u(V^\pi(s)).
\] (2.13)

This process is often referred to as scalarisation in the literature\[43\].
2. Background

Solutions Sets for Multi-Objective Decision-Making Problems

If the utility function of the user is not known, sets of candidate solutions to the problem must be found and presented to the user for selection. The most general set of possibly optimal policies and associated value vectors is the undominated set \( U(\Pi) \).

**Definition 4** The undominated set, \( U(\Pi) \), is the subset of all possible policies \( \Pi \) and associated value vectors for which there exists a possible utility function \( u \) with a maximal scalarised value:

\[
U(\Pi) = \left\{ \pi \in \Pi \left| \exists u, \forall \pi' \in \Pi : u(V^{\pi}) \geq u(V^{\pi'}) \right. \right\}. \tag{2.14}
\]

One problem with the undominated set is that it may contain an unnecessary number of policies, since several policies may be optimal for the same utility function. As a result, learning the full set may waste computational resources. To reduce the number of policies we can instead try to find a coverage set \( CS(\Pi) \).

**Definition 5** A set \( CS(\Pi) \) is a coverage set if it is a subset of \( U(\Pi) \) and if, for every \( u \), it contains a policy with maximal scalarised value, i.e.,

\[
CS(\Pi) \subseteq U(\Pi) \land \left( \forall u, \exists \pi \in CS(\Pi), \forall \pi' \in \Pi : u(V^{\pi}) \geq u(V^{\pi'}) \right). \tag{2.15}
\]

To further reduce the size of the solution set, we can reason about the types of utility functions that we are interested in. Since in MORL we are interested in increasing the value for each objective, we can assume that the utility functions used are monotonically increasing in all objectives \( [45] \).

**Definition 6** A monotonically increasing utility function, \( u \) adheres to the constraint that if a policy increases for one or more of its objectives without decreasing any of the objectives, the scalarised value also increases:

\[
(\forall i : V_i^{\pi} \geq V_i^{\pi'}) \land (\exists i : V_i^{\pi} > V_i^{\pi'}) \implies u(V^{\pi}) \geq u(V^{\pi'}). \tag{2.16}
\]

With this assumption about the utility function, the definition of the undominated set becomes the Pareto Front (PF):

**Definition 7** If the utility function \( u \) is any monotonically increasing function, then the Pareto Front (PF) is the undominated set \( U(\Pi) \):

\[
PF(\Pi) = \{ \pi \in \Pi \mid \exists \pi' \in \Pi : V^{\pi'} >_{P} V^{\pi} \}, \tag{2.17}
\]

where \( >_{P} \) is the Pareto dominance relation,

\[
V^{\pi} >_{P} V^{\pi'} \iff (\forall i : V_i^{\pi} \geq V_i^{\pi'}) \land (\exists i : V_i^{\pi} > V_i^{\pi'}). \tag{2.18}
\]
A minimal set of policies that generate the values of the Pareto front is a Pareto Coverage Set (PCS) \[45\].

If we further restrict the utility function to be a linear weighted sum of the objective values, the undominated set is the Convex Hull (CH) \[45\]:

**Definition 8** A linear utility function computes the inner product of a weight vector \(w\) and a value vector \(V^\pi\)

\[
u(V^\pi) = w^T V^\pi.
\] (2.18)

Each element of \(w\) specifies how much one unit of value for the corresponding objective contributes to the scalarised value. The elements of the weight vector \(w\) are all positive real numbers and constrained to sum to 1.

**Definition 9** The convex hull (CH) is the subset of \(\Pi\) for which there exists a \(w\) (for a linear \(u\)) for which the linearly scalarised value is maximal, i.e., it is the undominated set for linear utility functions:

\[
CH(\Pi) = \{ \pi \in \Pi \mid \exists w, \forall \pi' \in \Pi : w^T V^\pi \geq w^T V^{\pi'} \},
\] (2.19)

and its coverage set is the Convex Coverage Set (CCS):

**Definition 10** A set CCS(\(\Pi\)) is a convex coverage set if it is a subset of \(CH(\Pi)\) and if for every \(w\) it contains a policy whose linearly scalarised value is maximal, i.e., if:

\[
CCS(\Pi) \subseteq CH(\Pi) \land \left( \forall w, \exists \pi \in CCS(\Pi), \forall \pi' \in \Pi : w^T V^\pi \geq w^T V^{\pi'} \right).\] (2.20)

The solution sets for multi-objective reinforcement learning are illustrated in Figure 2.5. For efficient learning, it is important to identify which set of solutions is the best one for a specific user. This is further discussed in the next section.

**Axiomatic-Based versus Utility-Based Approaches**

Many works in multi-objective reinforcement learning have focused on axiomatic-based approaches, which assume that finding the Pareto front is the optimal solution \[43\]. This may seem reasonable in situations where the user’s utility function is not known, since the Pareto front contains an optimal value vector for any monotonically increasing utility function. However, the Pareto front may contain many, or possibly an infinite number of value vectors, making it computationally challenging or intractable to find all relevant solutions. Instead an approximation must be found. Such solution set approximations are evaluated using axiomatic-based metrics, which try to estimate how well a solution set recovers the Pareto front in terms of, e.g., spread, coverage, or distance.
Figure 2.5: Solution sets for multi-objective reinforcement learning [45].

One common axiomatic-based metric is the hyper-volume metric, which measures the hyper-volume in value-space that is Pareto-dominated by a solution set found by a MORL algorithm [173]. The hyper-volume is calculated as [45]:

\[
\text{HyperVolume}(CS, V_{ref}) = \bigcup_{\pi \in CS} \text{Volume}(V_{ref}, V_{\pi}),
\]

where \(\text{Volume}(V_{ref}, V_{\pi})\) is the volume of the hypercube spanned by the reference vector, \(V_{ref}\), and the vector in the CS, \(V_{\pi}\). The hyper-volume of the Pareto front is the maximum volume that can be achieved by any solution set, and thus represents an upper limit for learning algorithms. However, it is difficult to interpret the value of individual improvements of the volume, since it is not directly related to the utility of the user. Figure 2.6 illustrates the hyper-volume for a problem with two objectives, where two new solutions are added. Adding an undominated solution at the edge of the front leads to a greater increase in hyper-volume compared to adding an undominated solution in the middle of the front, even though it is not obvious which solution would be of greatest utility for the user.

Another commonly used axiomatic-based evaluation metric is diversity of the solution set [45]. The reasoning behind this metric is that a set of solutions that is diverse in terms of the multi-objective returns can provide useful solutions to a larger group of potential users, whose objective preferences may vary. One metric related to diversity is the crowding distance [26], which measures the distance between neighbouring solutions in value-space. If two solution sets have the same hyper-volume, the set whose value-vectors are more evenly spread over the front is considered the better one. A related metric is the sparsity metric [161]:

\[\text{HyperVolume}(CS, V_{ref}) = \bigcup_{\pi \in CS} \text{Volume}(V_{ref}, V_{\pi}), \]
2.6. Multi-Objective Reinforcement Learning

Figure 2.6: Example of a hyper-volume calculation, with a reference vector in blue, dominated solutions in black, and undominated solutions in red. The green solutions illustrate the effect on the hyper-volume by adding two new undominated solutions to different parts of the front [13].

\[
S_p(S) = \frac{1}{|S| - 1} \sum_{j=1}^{m} \sum_{i=1}^{|S| - 1} (\tilde{S}_j(i) - \tilde{S}_j(i + 1))^2, \tag{2.22}
\]

where \( S \) is the undominated set, \( \tilde{S}_j(i) \) is the \( i \)-th value in the sorted list for the \( j \)-th objective value in \( S \), and \( m \) is the number of objectives of the environment. If the coverage of the front is less sparse, there are more solutions for the user to choose from, and a higher achieved utility can be expected.

Instead of by default trying to learn the full Pareto front, the utility-based approach proposes a 5-step process (illustrated in Figure 2.7) to more efficiently learn a set of policies suitable for an identified user [13]:

1. Collect all a priori available information regarding a user’s utility.
2. Decide which type of policies are allowed.
3. Derive the optimal solution concept from the resulting information of the first two points.
4. Select or design a MORL algorithm that fits the solution concept.
5. When multiple policies are required for the solution, design a method for the user to select the desired policy among these optimal policies.

In the first step, the aim is to collect all available information about the user’s utility. This includes what type of utility function is suitable, e.g. linear or non-linear, as well as what parts of the value-space might be of most
interest to the user. Furthermore, it should be determined how the utility should evaluated. One possibility is to evaluate the expected utility of the return of single episodes, which is known as the Expected Scalarised Returns (ESR) optimality criterion \(^45\). Another possibility is to evaluate the utility of the expected return of multiple episodes, which is known as the Scalarised Expected Returns (SER) optimality criterion. \(^45\). The identification of user utility will provide valuable information about the domains where the learning agents will be used, which can help design the right method for learning and selecting policies.

In the second step, the type of policy to use is selected based on the knowledge gained about the user’s utility. For instance, deterministic policies may be preferable in safety-critical applications, while stochastic policies (e.g. stochastic mixture policies \(^141\)) may provide better performance in other settings.

In the third step, the combined knowledge of steps 1 and 2 is used to derive a suitable solution concept. If there is not a single, known utility function, then the gained knowledge about the user would ideally allow us to find the smallest solution set that contains all policies relevant for the user. Otherwise, the optimal solution set must be assumed to be the Pareto front, as when using an axiomatic-based approach.

In the fourth step, an algorithm matching the solution concept chosen in step 3 is selected or developed. If we have been able to identify a single utility function for the user, we can use an algorithm that aims to find a single policy that is optimal for the user’s utility function. If the user’s utility function is not known, or if it is expected to vary over time, an algorithm must be found that can learn multiple policies, each of which are expected to optimise the user’s utility in some future setting.

In the fifth step, if a policy set has been constructed, a method for helping the user select the policy with highest utility is designed. Then a selection phase is conducted, and the policy can then be executed in the target environment. If the policy set is large, the selection phase can be challenging.
In practice, steps 1 and 2, as well as steps 4 and 5, can be done in parallel in an iterative manner, as illustrated in Figure 2.7.

In addition to the process above, new evaluation methods have also been proposed to support a utility-based approach to MORL. Ideally, the solutions found should be evaluated in the operational environment, to see what utility they provide to the user. In a simulation-based training environment, one approach could be to first run automated tests to validate that the simulation is sufficiently close to reference scenarios from the real world, based on suitable metrics from the domain under study. Such tests could then be augmented by subjective evaluations with support from subject matter experts.

For situations where evaluations in the operational environment are not possible, Zintgraf et al. [172] propose the expected utility metric (EUM) and the maximal utility loss (MUL) for utility-based evaluations. EUM evaluates the utility of a solution set $S$ as the expected utility under a prior distribution over the user’s utility function [45]:

$$EUM = \mathbb{E}_{P_u} \left[ \max_{\pi \in S} u(V^\pi) \right]. \quad (2.23)$$

The distribution over utility functions $P_u$ is based on domain knowledge. In simulation-based training we could, for instance, model that we expect the proficiency of trainees to have a certain distribution over the classes beginner, intermediate, and advanced.

MUL evaluates the utility of a solution set $S$ by measuring the maximum loss in utility caused by using solutions from this set instead of using solutions from an optimal solution set, when using utility functions from the space of possible utility functions $U$ [45]:

$$MUL = \max_{u \in U} \left( \max_{\pi \in S^*} u(V^{\pi^*}) - \max_{\pi \in S} u(V^{\pi}) \right). \quad (2.24)$$

Since in real-world problems the optimal solution is often not known, MUL can be used as a relative metric to compare solution sets found by different algorithms.

### Algorithms for Multi-Objective Reinforcement Learning

When using the utility-based approach, the choice of algorithm depends on the knowledge about the users and their utility functions. Multi-objective reinforcement learning algorithms can be divided into single- and multi-policy algorithms. Single-policy algorithms require that the optimal utility function of the user is known, and can be used to guide the learning algorithm to a corresponding optimal policy. When modelling user utility with linear utility functions, we can use scalarised reinforcement learning [90] to learn policies. One way is to scalarise the MOMDP into an MDP, and then find a policy with single-objective methods. Another way is to learn in the MOMDP, but
calculate the scalar values of vector value functions, which can then be used in the normal update rules for standard single-objective value-based or actor-critic style reinforcement learning algorithms, such as DQN and DDPG, as well as for greedy action selection in value-based methods. One disadvantage of scalarised learning with linear utility functions is that we cannot find solutions in the concave parts of the Pareto front [144]. Instead we can try to find solutions in the convex coverage set. In some problems this may still result in an optimal solution, for instance when the problem’s objectives are expressed in monetary terms [45]. However, many real-world problems need to be modelled with non-linear utility functions [144]. For instance, when safety is considered the utility function may contain thresholds for acceptable risk-taking.

If the utility function is known, but not linear, things are not as straightforward. When value-based methods are used, a naive scalarisation of the MOMDP would violate the assumption of additive returns in the Bellman equation [114]. To address this, the observation space of the agent can be extended to include information about the rewards accumulated so far [30]. The current sum of rewards can then be added to the expected future return from the current state before applying the utility function to get the scalarised expected return. An alternative is to use policy-search algorithms, e.g. policy gradient algorithms, which do not rely on the Bellman equation [45, 101]. When using non-linear utility functions, we must also decide if we want to optimise for the utility of the expected outcome of multiple episodes (SER), or the expected utility of individual episodes (ESR). For the ESR setting, Roijers et al. proposed an Expected Utility Policy Gradient (EUPG), which extends Monte Carlo policy gradients to optimise for the utility of the episodic return [113, 129, 158]. The EUPG loss function is defined as:

$$L(\pi) = -\sum_{t} u(R_{-}^\tau + R_{+}^\tau) \log \pi_{\theta}(a|s, R_{-}^\tau, t),$$  \hspace{1cm} (2.25)

where

$$R_{-}^\tau = \sum_{t=0}^{\tau-1} \gamma^t r_t,$$  \hspace{1cm} (2.26)

is the return up until time step \(\tau\), and

$$R_{+}^\tau = \sum_{t=\tau}^{T-1} \gamma^t r_t,$$  \hspace{1cm} (2.27)

is the return from time step \(\tau\) to the horizon \(T\). The policy \(\pi_{\theta}\) conditions on the return achieved so far \(R_{-}^\tau\) and the current time step \(t\), to enable finding the future return \(R_{+}^\tau\) that will maximise the episodic utility of the user. More recently, Reymond et al. proposed a multi-objective categorical actor-critic (MOCAC) that improves upon EUPG by using a distributional critic to estimate a categorical distribution over multi-objective returns [108]. Hayes et
al. took a similar approach, and used distributional Monte Carlo tree search (DMCTS) to learn a distribution over the utility of the returns [47].

When the utility function is not known, multi-policy methods try to find a suitable solution set for the user. Multi-policy algorithms can be further divided into outer loop and inner loop approaches [45]. Outer loop approaches use an outer loop that iterates over utility functions to optimise for, and for each selected utility function calls a single-policy MORL algorithm as a subroutine to find the optimal solution. Inner loop methods directly learn a set of policies in parallel. The efficiency of outer loop algorithms can be improved by directing training towards the utility functions that are expected to improve the current approximate coverage set the most [90, 161], and by reusing information from past learning, for instance using learned policies as a starting point for learning with a new utility function [93, 90, 3, 163, 107].

Inner loop methods typically use methods from single-objective reinforcement learning as a basis, and modify those methods to make it possible to update several policies based on a single interaction with the environment. For instance, Pareto-Q-Learning stores a set of achievable Pareto optimal values instead of a single Q-value [145]. The method was extended to problems with high-dimensional states in Pareto DQN [109]. Yang et al. proposed to apply envelope updates to a multi-objective DQN, to update the network for multiple linear utility functions at once instead of using scalarisation with a single utility function.

2.7 Multi-Objective Multi-Agent Reinforcement Learning

Just as reinforcement learning can be generalised to multi-objective reinforcement learning by using vector rewards, multi-objective multi-agent decision-making problems can be formalised by extending the existing models for single-objective multi-agent decision-making, e.g. stochastic games, to use vector reward functions. The most general model is the Multi-Objective Partially Observable Stochastic Game (MOPOSG).

Definition 11 A MOPOSG is defined by a tuple \( \langle S, A, T, R, \gamma, \mu \rangle \), with \( n \geq 2 \) agents and \( d \geq 2 \) objectives, where [13]:

- \( S \) is the state space
- \( A = A_1 \times \cdots \times A_n \) represents the set of joint actions, with \( A_i \) being the action set of agent \( i \)
- \( T : S \times A \times S \rightarrow [0,1] \) is a probabilistic transition function
- \( R = R_1 \times \cdots \times R_n \) are the reward functions, where \( R_i : S \times A \times S \rightarrow \mathbb{R}^d \) is the vector reward function of agent \( i \) for each of the \( d \) objectives


2. Background

- $\gamma \in [0,1)$ is a discount factor indicating the importance of immediate and future rewards respectively
- $\mu : S \rightarrow [0,1]$ is a probability distribution over initial states

As in the multi-objective single-agent setting, the best solution of each agent is determined by the utility of its multi-objective return. In the taxonomy of multi-objective multi-agent decision-making proposed by Radulescu et al. [103], the solution concepts Coverage Sets, Equilibria and Stability Concepts, and Mechanism Design are mapped to settings with different allocations of reward and utility functions to the participating agents.

When agents receive a team reward and all agents share the same utility function, there are no conflicts among the agents, and the solution concepts are related to different types of coverage sets, as in multi-objective single-agent decision-making. Coverage sets can also be a suitable solution when the agents of the system receive different utilities. In settings where agents have a team reward, but individual utility functions, negotiation could be used to select a solution from a coverage set. In settings where agents have individual utility functions and also receive individual rewards, coverage sets can be used as best responses when interacting with other agents.

Game theoretic equilibrium solutions, such as Nash equilibrium [92] and correlated equilibrium [8], can be used as solution concepts when agents have individual utility functions and receive either team rewards or individual rewards. In the multi-objective setting equilibrium solutions are affected by the chosen optimisation criterion (optimising for scalarised expected returns or expected scalarised returns), and a Nash equilibrium may not even exist for normal form games (unlike in single-objective normal form games) [104].

In some settings, the goal is to optimise a system wide utility function $W$, which considers the utilities of all agents, each of which in turn are based on the reward and utility functions of the agents:

$$u_w = W(u_1(V_1), \ldots, u_N(V_N)).$$

In this setting, mechanism design can be used to design rewards that encourage the agents to be truthful about their individual utility functions, and to learn policies that optimise social welfare according to $W$.

2.8 Improving the Efficiency of Learning

In complex domains, learning new policies may require many iterations. This may delay updates of systems, and user needs may not be met. In adaptive simulation-based training, we want to continuously adapt the behaviour of synthetic agents in the simulation, so that it matches trainees’ current training needs. Therefore the efficiency of learning algorithms used is important.
2.8. Improving the Efficiency of Learning

Efficiency can be improved by, e.g., modifying the reward signal or updating the learning approach.

**Potential-Based Reward Shaping**

In some environments, learning efficiency is negatively affected by sparse rewards. This is the case in many air combat scenarios, where many time steps must be completed to solve tasks, and a reward may only be given once the task is completely solved. One way to improve efficiency in such settings is to add additional components in the reward signal, for instance rewards for solving significant sub tasks. Such *reward shaping* can help the learning agent explore the most relevant parts of the environment. As an example, $Q$ learning with reward shaping $F(s_t, a_t, s_{t+1})$ can be defined as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha (r_t + F(s_t, a_t, s_{t+1}) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)).$$ \hspace{1cm} (2.29)

One possible risk with reward shaping is that a bias may be introduced in the policy, i.e. the agent is prevented from finding an optimal policy for the original problem. It has been shown that the only reward shaping that is guaranteed to not have this effect is potential-based reward shaping [94]. Potential-based reward shaping defines the shaping reward as the difference in potential of successive states:

$$F(s_t, a_t, s_{t+1}) = \gamma \phi(s_{t+1}) - \phi(s_t)$$ \hspace{1cm} (2.30)

Here $\phi(s)$ is the potential of a given state, and $\gamma$ is the discount factor of the MDP.

As an example of potential-based reward shaping, consider the simple scenario presented in Figure 2.8, where we would like the agent to navigate from the start position to the target. Due to the distance between the start position and the goal state, it may be difficult for the agent to find the target if a sparse reward signal is used. If we instead define a potential based on the distance to the target, the agent can be given continuous feedback regarding its progress towards its goal.

Mannion et al. extended the study of reward shaping to multi-objective reinforcement learning, and demonstrated that for linear utility functions policy invariance was guaranteed for potential-based reward shaping as well [87]. In addition, it was demonstrated that this was true whether the shaping was applied before or after scalarisation.

It is worth noting that though potential-based reward shaping is guaranteed to not introduce a bias in the policy, there is no guarantee that it will help the training to converge faster.
2. Background

Figure 2.8: Potential-based reward shaping for a navigation task, where the agent is rewarded for moving to states with higher potential.

Curriculum Learning

Instead of modifying the reward signal, it may be possible to modify the learning environment, so that the agent is exposed to successively more challenging tasks as its performance is improved. This approach is similar to the one taken in adaptive training of humans, and it is referred to as Curriculum Learning when training reinforcement learning agents. With a suitable design of the curriculum, the agent will be able to continually update its policy as it is transferred from one task to another. As in adaptive training systems for humans, the curriculum should ideally include a measurement of the agent’s performance and automatically update the environment to improve it.

An example of curriculum learning for air combat simulation is illustrated to the left in Figure 2.9. In this example an agent should navigate its aircraft from a start position to a target, while avoiding threats in the form of air defence systems. For this scenario, a curriculum could be designed, that successively adds more threats to the environment, requiring more attention and precise navigation from the agent. In the figure, Lesson 1 is an environment without threats, while Lesson 2 and Lesson 3 each add an additional threat zone that the agent must consider.

Another approach, which is also suitable for navigation tasks in air combat scenarios, was proposed by Florensa et al. Instead of modifying the structure of the environment, the distribution over initial states, \( \mu(S) \rightarrow [0,1] \), can be adjusted over time to make it increasingly challenging to reach a goal state. An example of this approach is illustrated to the right in Figure 2.9. In this scenario, the probability of the aircraft’s initial state being far from the goal state, represented by the target, increases over time. This requires the pilot to avoid more threat areas, which increases the challenge of the task.
Andrychowicz et al. proposed Hindsight Experience Replay (HER), a novel approach for curriculum learning that works by adjusting the experiences of learning agents for improved efficiency. The approach is aimed at multi-goal reinforcement learning environments, where the input to the policy of an agent is not only the current state of the environment, but also a goal state that the agent should reach. In the air combat domain the goal could be a specification of a target that a synthetic pilot should attack and destroy. When HER stores experiences in a replay buffer, the actual goal specified in the episode is augmented with a set of additional goals, e.g. possible goal states that were actually achieved by the agent. When the agent’s policy is trained with samples from the replay buffer, it results in an implicit curriculum, since simple as well as difficult goals are experienced in the replay phase. The approach was evaluated with good results on robotic arm manipulation tasks, where agents received only binary rewards, which indicated if the task had been solved or not.
Part I

Domain Analysis and Concept Design
In this chapter, we study learning agents from a user perspective, with support from experienced fighter pilots. The goal is to learn more about how intelligent agents could be used to automate some of the tasks performed by human training providers. First, we set the context by performing an analysis of the domain of simulation-based training. The analysis is conducted from the perspectives of instructors, role-players, and trainee pilots respectively. The purpose of the analysis is to identify constraints imposed on training providers when using different types of simulation resources, and to model the patterns of decision-making a synthetic agent must be capable of if it is to replace human role-players in air combat scenarios. Then, we conduct a user study consisting of repeated interviews and a written survey, with the purpose of finding out what experienced pilots consider important agent capabilities and characteristics in different phases of air combat training. This chapter is based on Paper VI [67].

3.1 Constraints Affecting Instructors

In interviews with pilots, we tried to identify constraints that they currently face when acting in the role of instructor. Based on these interviews, Figure 3.1 shows a subjective measure of the relative importance of different types of constraints, for three different types of training simulations: Live, Virtual, and Constructive. While considering these constraints, the challenge of the organisation that provides training is to find an optimal trade-off between
3. User Study

Figure 3.1: Subjective ranking of constraints affecting training for different types of simulation resources.

training value and training availability. We discuss the identified constraints further in the text below.

Live training simulation provides the highest possible fidelity in terms of interaction with the aircraft and its subsystems. However, in the Live setting, training is highly affected by aspects of the physical world. For instance, the availability of vehicles and other types of systems may not be sufficient to realise complex scenarios. In particular, a military organisation may not have access to systems that have the same performance and characteristics as those that are used by the enemy. Furthermore, operation of physical vehicles, e.g. aircraft, is highly expensive, which limits the amount of training that can be delivered in this setting.

Training in the Live setting is also constrained by the limited availability of air space, as well as safety regulations, which makes it difficult to realise scenarios with many entities, who are operating over a large geographical area. A large number of support personnel may also be required to plan and conduct such exercises. In addition, when acting in the open secrecy may be compromised, and there is a risk that systems’ performance and tactics are revealed to opponents.

By using ground-based, Virtual simulators, the constraints imposed by the physical environment are lifted, and training delivery becomes easier. Still, there is a considerable cost related to populating complex scenarios with a large number of high-fidelity simulators. Constraints regarding model fidelity increase in this setting, in particular for within visual range (WVR) combat, where the effects of, e.g., g-forces is an important factor for pilot performance.
To populate scenarios with only Virtual participants, some humans must act on the opponent’s side. The training value of acting on the opponent side depends on the credibility of the pilot stations used, i.e. how well they represent a real aircraft. If the pilots that act as opponents use pilot stations with high credibility to play their role in the scenario, they can learn to understand the opponent’s systems and tactics, which is valuable. However, if the simulators used for this type of role-play do not have sufficient credibility, e.g. highly simplified user interfaces, the training value will be lower.

Constructive simulation makes it possible to realise large scenarios, populated by synthetic entities, which can replace human role-players. This reduces the need for physical resources, so that only the computation hardware for running the simulation software is required. Instead, the constraints are shifted to the fidelity of the simulation models, and the available offline support for building the models, as well as the simulation scenarios. In particular, it becomes challenging to construct behaviour models for the synthetic entities, and adapting models to the training needs of individual trainees. Since the expertise required for such tasks may not be available locally, at each training facility, the turn-around time for updating training contents may be long. Instead, it may be necessary to have scenario operators manually control the flow of the tactical scenario to some extent.

In interviews with experienced pilots, we discussed to what extent agents could currently be used to provide high quality training, and what challenges instructors were currently facing. Typically, instructors are provided with synthetic entities and behaviour models from their support organisation. However, these models may not fit all relevant training cases, especially as time passes, and aspects of the operational environment change. Therefore, it would be good if instructors could adapt training contents on their own, without the support of simulator engineers. However, instructors feel that this is difficult when using the tools that are currently available for behaviour modelling. This is not surprising, since the construction of behaviour models for multi-agent systems is a highly challenging task, and a very active area of research. When simulator engineers must be involved, the turn-around time increases. It is also challenging to translate human domain expertise to model parameters that engineers can base their implementations upon.

In the interviews with pilots, they expressed that at present time, the highest training value is achieved when using agents as synthetic opponents. This reduces the need for support personnel, who do not receive training, to participate in training scenarios. It also reduces the need for expensive equipment, e.g., aircraft or high-fidelity simulators. However, handcrafted behaviour models often result in behaviour that comes across as scripted, static, and predictable. This may be acceptable for basic training, where pilots learn how to operate the aircraft and its tactical systems. However, for advanced tactical training the requirements are higher. To get the variation required in a stimulating learning environment, a lot of manual work and time...
must be invested, and the cost of keeping in pace with training needs may be high. By using machine learning, it could become possible to construct behaviour models that continually adapt to changes in the training environment, e.g., encounters with new trainees, introduction of new aircraft systems, and changes in trainees’ tactics. Machine learning could also help simplify the interfaces of the tools used to build and control agent behaviour, so that explicit programming were no longer required. Instead, instructors could use their domain knowledge to specify the goals and characteristics of the agents, in a similar way as learning goals and evaluation metrics are specified for human pilots. Data-driven methods could also provide objective evaluations of human pilots, on a machine-readable format, which could support automated adaptation of simulation contents, so that training scenarios are always in pace with training needs.

3.2 Human-Machine Interaction in Air Combat

In this section, we study aspects of human-machine interaction in air combat scenarios. The aim is to illustrate to what extent perception, decisions, and actions are supported by the automation of the aircraft or pre-planned procedures, and which parts of aircraft control that must be handled by the pilot alone. This information gives insight regarding requirements that must be fulfilled by synthetic agents that are to replace human pilots in training scenarios, and how to design the interface between the agent and the aircraft model, including its tactical systems. When creating synthetic pilots, the high-level tactical decisions made by human pilots will be the most challenging ones to model using AI. Therefore, the information available to support human decision-making at this level of abstraction should also be incorporated in AI algorithms to maximise their performance.

In support of our study, we use the Joint Control Framework Score (JCF-S) notation \[85\]. JCF-S is intended to support modelling of temporal aspects of human-machine interaction, at different levels of autonomy in cognitive control (LACC). The levels are summarised in the list below.

1 PHY The *Physical* level, which shows constraints related to physical actions.

2 IMP The *Implementation* level, which shows constraints related to implementation properties.

3 GEN The *Generic* level, which provides generic plans for common situations.

4 VAL The *Value* level, which handles trade-offs among the system’s objectives.
3.2. Human-Machine Interaction in Air Combat

5 EFF The Effect level, which deals with the system’s purpose and goals.

6 FRA The Framing level, which identifies the situation and context for control.

Levels 1 and 2 determine HOW control is realised, levels 3 and 4 deal with WHAT is done, and levels 5 and 6 are related to WHY the system exists. To model the joint control of human and machine, perception points (PP), decision points (DP), and action points (AP) are placed on six timelines, each of them representing one of the LACC levels. As a result, a pattern of the control loop of the joint system emerges. When agents are used as synthetic pilots in training scenarios, they should display a similar decision-making pattern as human pilots.

To illustrate how the capabilities of the pilot’s tactical control loop (Observe-Orient-Decide-Act) are mapped to different levels of cognitive control, we study the engagement of two pilots in offensive and defensive counterair operations, i.e., the quest for a favourable air situation, air superiority, or air supremacy. This is an example of beyond visual range (BVR) air combat, where pilots use long-range sensors and weapons. For this study, we use the scenario illustrated in Figure 3.2 to set the context. In this scenario, two aircraft are flying a Combat Air Patrol (CAP) directed towards the south, to protect their assigned Fighter Area Of Responsibility (FAOR), which is illustrated by the large circle in the figure. The FAOR contains three high-value assets, which are illustrated by the smaller circles in the figure. Approaching from the west are two hostile aircraft, which intend to perform an opportunistic attack on the high-value assets, and must therefore first deal with the defending aircraft of the CAP. We assume that the defensive fighters are trainees, while the offensive fighters are human role-players, who try to support the training of the trainees. They do this by adapting their behaviour to, e.g., provide a suitable level of challenge for the trainees.

The engagement is modelled using the JCF score notation, and the result is illustrated in Figure 3.3. To simplify the notation we only present the engagement of two of the aircraft in the scenario.

The timeline of the scenario is divided into four sections (I-IV), where significant events occur. The behaviour of the defending pilot is presented in the score in the upper part of the figure (labelled BLUE), and the behaviour of the attacking pilot is presented in the score in the lower part of the figure (labelled RED). At the top of the figure, the geometry between the two aircraft in different sections of the scenario is illustrated.

In section I of the scenario, a hostile RED aircraft is approaching the FAOR of the opposing BLUE aircraft. The approach is carried out according to a pre-planned procedure (AP on level 3 GEN). The pilot of the BLUE aircraft is informed by the decision support system that it is in the radar field-of-view of the RED aircraft, and uses the head-down displays (HDDs)
3. User Study

Figure 3.2: Hostile entities approaching a Combat Air Patrol (CAP) along the red route. The aircraft of the CAP are protecting the Fighter Area Of Responsibility (FAOR) in blue, where high-value assets in yellow are located.

Figure 3.3: Air combat training score for blue (in solid blue circle at top of figure) and red (in dashed red circle at top of figure) forces in a counterair operations scenario.

In section II of the scenario, the pilot of the BLUE aircraft once again refers to the HDDs, to assess how to best deal with the threat (PP on level 1 PHY). The decision is then made that the most valuable course of action is to engage the target (DP on level 4 VAL). After the decision has been communicated to update himself regarding the scenario geometry (PPs on levels 4 VAL and 1 PHY respectively). He then makes a decision regarding the current threat level (DP on level 6 FRA).
to the tactical air unit (AP on level 1 PHY), the pilot proceeds with target engagement according to doctrine (AP on level 3 GEN).

In section III of the scenario, the pilot of the RED aircraft is informed by the decision support system that it is in the radar field-of-view of the BLUE aircraft, which it is tracking (PP on level 4 VAL). The pilot considers desirable effects related to tactical mission goals as well as trainees’ training goals (DP on level 5 EFF), and decides to proceed into the BLUE aircraft’s FAOR, with the hope of attacking a high-value asset (DP on level 4 VAL). In the meantime, the pilot of the BLUE aircraft observes that the RED aircraft is now within range (PP on level 4 VAL), and decides to fire a missile (DP on level 4 VAL followed by AP on level 1 PHY).

In section IV, after firing the missile, the pilot of the BLUE aircraft guides it towards the target according to doctrine, until handover of control to the missile when it is close enough to open its target seeker and lock on the target aircraft (AP on level 3 GEN). The pilot of the RED aircraft is informed by the decision support system that there is an incoming missile (PP on level 4 VAL), and performs an evasive manoeuvre to avoid the threat (DP followed by AP on level 3 GEN).

We can see that pilots are supported by refined, abstract information, provided by the decision support system of the aircraft, to form their situational awareness. We can also see that several actions are pre-defined to handle a certain situation, and have a temporal extension, e.g., target approach procedures, missile guidance procedures, and evasive manoeuvres. Finally, decisions on how to handle the situations that occur are often taken at the higher levels of cognitive control, were full automation may not currently be available. Therefore, pilots still play a vital part in the outcome of missions. They must have the capability to comprehend the situation, to identify and rank potential threats and targets. Then, when acting upon their situational awareness, pilots must carefully choose how to use the tactical systems of the aircraft to balance multiple objectives. We will use the LACC further when discussing agents' decision-making in the following sections of the dissertation.

3.3 User Needs in Simulation-Based Training using Agents

In this section, we present the results of a user study, which aimed at identifying how synthetic agents could help make simulation-based pilot training more efficient and effective. We discuss which capabilities and characteristics agents are expected to have, from the perspectives of trainee pilots as well as instructors.
3. User Study

Organisation of the Study

The study consisted of repeated user interviews, and a follow-up written survey. The participants of the interviews and the survey were experienced fighter pilots from the Swedish Air Force, and experienced test pilots from Saab Aeronautics. Three pilots participated in the interviews, while twenty-five pilots participated in the survey. Since this sample of pilots is quite small, the views expressed may not be fully representative of the whole population of pilots, but hopefully the feedback given by the participants can help identify important aspects related to the design of agents that are intended to act as synthetic pilots in training scenarios.

The goal of the interviews was to allow pilots to describe current challenges in pilot training, and possible areas of improvement. In particular, the focus was on ways to automate training delivery to a higher degree using intelligent, learning agents, to reduce the dependency on support personnel such as role-players and scenario operators, and to improve the availability of high-quality training while reducing cost. Participants were initially asked to share their thoughts on training goals, training approaches, and training media, to give an unbiased overview of how training is currently conducted. Thereafter, the interviewers took a more active part, to identify the achievable training value when using agents in place of human role-players, to learn about challenges related to constructing training scenarios when using agents, and to discuss what role learning agents could play in simulation-based training systems in the future.

The interviews with pilots revealed a set of important factors that would need to be considered in the design of synthetic agents. For the written survey, based on the information gathered through the conducted interviews, a number of statements regarding desirable agent capabilities and characteristics were presented to the participants. They were asked to rate to what degree they agreed with the statements, for three different types of training: Basic Training, Tactical Procedure Training, and Mission Training. The intention was to identify how user needs differed for these three types of training. We define the categories of training as follows:

- **Basic Training**: In the Basic Training phase, a pilot with previous experience in flying a different type of aircraft, or a different edition of an aircraft, is trained in basic flight manoeuvres and system operation.

- **Tactical Procedure Training**: In the Tactical Procedure Training phase, a pilot is trained in using, e.g., tactical sensors, data links, and weapon systems in typical combat scenarios.

- **Mission Training**: In the Mission Training phase, pilots are trained to cooperate in teams to carry out typical operational missions.
3.3. User Needs in Simulation-Based Training using Agents

The statements presented to respondents were divided into three categories: *Types of Agent Behaviour, Human-Agent Interaction, and Agent Behaviour in Training Scenarios*. Respondents were asked to give a score in the range one (low importance) to ten (high importance) for each statement, and they also had the possibility to add additional comments in free text. The results of the survey are presented as box plots (created using Matplotlib) for each category of training. In these plots, the horizontal line of a box represents the median value, the upper and lower edges (hinges) of a box represent the upper and lower quartiles, and the lines (whiskers) protruding from a box represent the upper and lower extreme values. Circles outside the whiskers represent data points that are classified as outliers, if they exist. The notches surrounding the medians of the boxes can be used to judge the significance of the difference between two median values. If the notches of two boxes do not overlap, then the confidence level of the difference is roughly 95%. If a notch would go outside of a box, protruding notches (with a “flipped” appearance) are plotted, which indicates a skewed distribution of values for that category. Median values are connected with a line for each statement, to help identify trends.

Desirable Agent Capabilities and Characteristics

For the category of *Types of Agent Behaviour*, the following statements were presented to respondents for rating:

- **Deterministic**: It is important that synthetic tactical entities can be given a deterministic behaviour.
- **Advanced**: It is important that synthetic tactical entities can display advanced tactical behaviour.
- **Doctrinal**: It is important that synthetic tactical entities can act according to doctrine.

The aim of this category of statements was to investigate the importance of different types of agent behaviour in different types of training. The scores given by respondents, which indicate to what degree they agree with the statements, are presented in Figure 3.4.

Regarding the ability to assign agents a Deterministic behaviour, we can see that this is important in all phases of training, although the scores vary more for mission training. The scores for Advanced behaviour increases as we move from Basic Training, where the importance is modest, to Mission Training, where the importance is very high. Here, there is some variance in the scores for Basic Training and Tactical Procedure Training, while the pilots are in high agreement concerning the importance of Mission Training with advanced synthetic opponents. Finally, the importance of Doctrinal behaviour
3. User Study

Figure 3.4: Importance of different types of agent behaviour.

shows a similar pattern as Advanced behaviour, becoming more important as we move towards Mission Training, although it is fairly important already for Basic Training. As for Advanced behaviour, the variance in the scores is higher for Basic Training and Tactical Procedure Training than for Mission Training.

In interviews, pilots expressed that in the initial phases of training, the requirements on synthetic opponents are rather modest. In this phase of training, it is important that the behaviour of synthetic opponents is predictable. The most important thing is to be able to create well defined, deterministic scenarios. For instance, when learning the functions and controls of a new sensor system, it may be distracting if opponents behave in an unpredictable manner. Instead, entities may move along predefined trajectories, or the positions of vehicles, including the trainee’s own aircraft, may be frozen. In many scenarios, there are synthetic entities that are primarily used as background noise, and it is then desirable that they can perform simple tasks such as start and landing. For entities that play a tactical role in the scenario, there are also well established, standard manoeuvres that they are expected to be able to carry out [120], such as straight flight, turning while keeping the radar within gimbal limits to maintain a target track, and performing pincer manoeuvre (attacking an enemy formation from both sides).

As the training progresses, more advanced synthetic pilots, who can take defensive as well as offensive roles, are required to realise scenarios that allow trainees to develop their tactical proficiency. One pilot reasoned that a good base requirement for entity behaviour is the ability to respond, in a believable way, to all orders available on the aircraft tactical data link. Furthermore, as explained by the participants in the study, providing entities that display
advanced tactical behaviour is a necessary, but not sufficient condition. It is also required that synthetic pilots can follow a certain doctrine when acting in training scenarios, to prepare trainees for a variety of potential adversaries. That is, it must be possible to model important aspects of established procedures for completing a certain type of mission, e.g., standard manoeuvres, definitions of high-value targets, and directions for when to attack or retreat. Such a modelling capability is a natural component of Mission Training, which is supposed to support preparations for specific missions. This is also supported by the results of the survey. This means that when behaviour models are developed using machine learning techniques, there must be a way to infuse domain knowledge in the learning process, so that the resulting behaviour fulfils rules encoded in a specific doctrine.

For the category of Human-Agent Interaction, the following statements were presented to respondents for rating:

- **Challenging Opponent**: It is important that synthetic tactical entities can act as challenging opponents (e.g. by discovering and exploiting flaws in the human trainees’ tactics and execution).
- **Wingmate**: It is important that synthetic tactical entities can act as wingmates of human trainees, with intelligent behaviour.
- **Voice Communication**: It is important that a synthetic tactical entity that acts as wingmate can communicate with human trainees through radio voice communication.

The aim of this category of statements was to investigate the importance of having agents act in different types of roles in different types of training scenarios, as well as the importance of voice interaction with agents. The scores given by respondents, which indicate to what degree they agree with the statements, are presented in Figure 3.5.

Regarding the ability of agents to act as Challenging Opponents, we can see a quite wide range of scores from Basic Training to Mission Training. The median score for Basic Training is low, while the median scores for Tactical Procedure Training and Mission Training are high. There is quite large variance in the responses for the two simpler categories of training, while respondents are more in agreement for the category of Mission Training. Being able to form teams with a mix of humans and agents, where agents act as intelligent Wingmates, is considered important for Tactical Procedure Training and Mission Training, but less important for Basic Training, where simpler scenarios are often used for training, and the main goal of the trainee is to become a proficient wingmate. The importance of Voice Communication received scores in the middle of the range, and with high variance.

In Basic Training, having too challenging opponents may make it difficult to focus on learning how to, e.g., operate sensor and weapon systems. Instead,
as noted previously, opponents may be configured to move along predefined routes, while acting according to predefined, predictable rules. For Tactical Procedure Training and Mission Training, having Challenging Opponents is essential, to evaluate the performance of trainees, as well as to validate the effectiveness of developed tactics. Pilots reasoned that if agents had a learning capability, they could identify flaws in human-developed tactics, and learn to exploit those flaws.

To be challenging opponents, agents need to possess similar capabilities as human pilots. Among other things, key to winning the light is to coordinate with your teammates, achieve high time on station, and to detect others while not being detected yourself. Together with teammates, synthetic pilots need to select good formations, maintain a favourable scenario geometry in relation to enemies (e.g., position, altitude, and movement), and keep enemies outside their maximum relative range of firing missiles at high-value targets while the members of the team themselves move into range to fire their missiles. It is important to maintain pressure on the enemy and cover a lot of surface (depth and width). Synthetic pilots should also be able to learn to identify weak opponents, and target them for attack in coordination with teammates. They must carefully consider when to engage an enemy based on its value and threat level, so as to not take unnecessary risk, or waste fuel and missiles. In a similar way, when using sensor or electronic warfare systems, emission management must be considered to balance the chance of detecting opponents while avoiding being detected by enemies.

Since pilots do not operate on their own in real-world missions, support for team training is of utmost importance, as indicated by the scores from the survey. In interviews, pilots expressed that having synthetic pilots that are
3.3. User Needs in Simulation-Based Training using Agents

intelligent enough to act as Wingmates of trainees is valuable, since it makes it possible to train as you fight even when there are not enough human pilots available to populate complex scenarios. At a minimum, self-paced training with 2-vs-2 fighters and a strike force should be supported. This requires that synthetic pilots can learn to understand the intentions of trainees, as well as their own role in the mission. In basic training, the major goal of the trainee is to become a proficient wingmate, and the availability of a synthetic formation leader to support self-paced training may be of higher importance than a synthetic wingmate. Pilots also expressed that even if synthetic wingmates had human-level intelligence, it would still be important to conduct advanced tactical and mission training in units populated by the other members of your air force wing, since these are the people you would cooperate with in real-world combat. This is reflected by the results of the survey, where scores for having a synthetic wingmate drop slightly for mission training compared to tactical procedure training. For mission training, having challenging opponents instead receives higher scores.

For success in air combat, it is important that the members of a unit coordinate their actions well. Therefore, some level of communication capability among human and synthetic agents may be required. The need for voice communication within mixed teams of human and synthetic agents was included in the survey since it is a rich form of communication, which in general may be challenging to realise in a believable way for synthetic agents. However, pilots reasoned that in air combat the information exchange over radio channels is often of a simple form, following a predefined protocol. Using domain knowledge makes it possible to predict what types of interaction will occur, which helps when building models for the speech understanding and speech synthesis of synthetic pilots. Pilots also argued, that in many situations they know how to respond to teammates actions without communication, since the team is trained in executing coordinated manoeuvres. However, realising such a capability in a synthetic pilot may be challenging.

Populating training scenarios with mixed teams of human and synthetic agents can make training more efficient and effective. When using machine learning to build synthetic pilots, learning behaviour that supports interaction with humans is important, but also challenging, since during learning agents typically act in a simulation where no humans are present. Therefore, learning methods that result in behaviour that generalises to diverse environments and scenarios are important.

For the category of Agent Behaviour in Training Scenarios, the following statements were presented to respondents for rating:

- **Agent Performance**: It is important that synthetic tactical entities have realistic performance (e.g., do not always execute weapon delivery and evasive manoeuvres perfectly).
3. User Study

• **Element of Surprise**: It is important that there is an element of surprise in the tactical scenario (i.e., the scenario does not play out in the exact same way in each run).

• **Behaviour Explainability**: It is important that it is possible to explain the behaviour of synthetic tactical entities in debriefing sessions (e.g., why a missile was fired in a certain situation).

The aim of his category of statements was to investigate the importance of different agent characteristics in the context of a training scenario. The scores given by respondees, which indicate to what degree they agree with the statements, are presented in Figure 3.6.

For the category of Agent Performance, we can see that there is high variance in the scores, but the median is at the lower half of the range for Basic Training. For Tactical Procedure Training and Mission Training, on the other hand, the importance of having realistic performance is high. The importance of having an Element of Surprise and variation in training scenarios increases as we move from Basic Training to Mission Training. Behaviour Explainability is of high importance in all types of training.

In interviews, pilots argued that it is important that it is possible to adjust the agents’ performance to suit specific trainees and training scenarios. This imposes requirements on the algorithms used for learning the behaviour models of agents, requiring them to learn models that can be adjusted in a similar way as human role-players can be instructed how to act in a training scenario. As an alternative, the learning algorithms should be efficient enough for learning several behaviour models, each with different characteristics. In addition, in real-world air combat even experienced pilots will make
mistakes, so it is important that this happens in training scenarios as well. As one pilot said, the learning agents should ideally be able to act as the perfect pedagogical instructor, adapting their behaviour to the current training needs of trainees. Synthetic pilots should not be perfect (e.g., performing perfect evasive manoeuvres or missile delivery); they should make similar mistakes as human opponents would, so that trainees can learn to take advantage of such mistakes. When using human role-players for training, these will try to adapt to the proficiency level of the trainees, and then sometimes make small, intentional mistakes, which the trainees are expected to exploit.

Training scenarios with variation are important for advanced tactical training, so that trainees do not simply learn how the scenario plays out each time, and base their decisions on that information. Variation also makes it more difficult for trainees to exploit possible deficiencies in the behaviour models used to control synthetic pilots. The variation can be realised in different ways. As mentioned before, one way is to use stochastic instead of deterministic policies. Another, more expressive way is to introduce explicit variations in the scenario, e.g., by varying the goals of agents, by adapting the way in which agents try to achieve those goals, for instance by specifying different rules of engagement, and by varying the characteristics of the agents that populate the training scenarios, such as their proficiency, aggressiveness, and level of risk-taking.

Training sessions are typically concluded with a debriefing session, where the outcome of the training scenario is discussed to determine what went well, what went less well, and areas for future improvement. In these sessions it is valuable if the decision-making process of the agents participating as synthetic role-players is transparent, so that the decisions made at key points in the scenario can be understood by the human participants. When using traditional techniques for constructing behaviour models, e.g., scripts, state machines, and behaviour trees, tools for analysis of behaviour can be constructed by extracting suitable information from those models. For learning agents that use neural networks to represent the decision-making policy, this process becomes more challenging, since neural networks are black-box models trained with data driven methods. This is true for deterministic as well as stochastic policies.

3.4 Summary

This chapter presented a user-centred analysis of introduction of intelligent, learning agents in simulation-based pilot training systems. First, we discussed constraints that instructors must deal with when providing training with Live, Virtual, and Constructive simulations, and how synthetic agents could remove some of the constraints if shortcomings of current agent technologies were addressed. Then, a model of pilots’ decision-making in counter air op-
3. User Study

A user study was constructed to identify what decision-making patterns should be supported by agent designs. Finally, interviews and a survey were used to identify requirements on agent capabilities and characteristics in different phases of pilot training.

We identified the potential of learning agents to improve the capabilities of constructive simulation, making it possible to use less live and virtual simulation resources while still providing effective training scenarios to trainees. We also noted that improved user interfaces could allow subject matter experts with little or no expertise in artificial intelligence to construct high quality agents, reducing the need for support from simulator engineers, and making it possible to reduce the turnaround time for updating the system according to training needs.

Feedback from users indicate the importance of having agents whose behaviour can be adapted to different types of training, as well as to trainees with different training needs. In addition, it is important that agents can act in different roles in scenarios with competition between teams of agents, such as opponent, group leader, and wingmate. Finally, being able to explain the behaviour of synthetic agents is important to support analysis in debriefing sessions. Current methods for developing behaviour models for synthetic pilots struggle to meet these requirements. For instance, users feel that constructive entities can not act in leading roles in air combat scenarios, fail to cooperate efficiently with human trainees, and lack the ability to respond to orders in a credible way \[1, 106, 159\]. Improved behaviour models could therefore increase the training value of LVC simulation.
This chapter investigates how the performance of agents learning coordinated navigation in simplified air combat scenarios depends on the choice of agent design and learning approach. Different designs of the agent’s action space, observation space, and reward function are more or less suitable for meeting the different requirements in user-adaptive training identified in Chapter 3. Being able to learn efficiently with all relevant designs is therefore important. We first briefly discuss the need for coordination and the motivation for studying coordinated navigation in Section 4.1. Then the method for the evaluation is presented in Section 4.2, including the learning algorithm and simulation environment used. Finally, the experimental evaluation is presented in Section 4.3. This chapter is based on Paper II [71], Paper III [72], Paper V [68], and Paper VI [67].

4.1 The Need for Coordination in Air Combat Simulation

As discussed in Chapter 3, a vital component of air combat is the capability of pilots to coordinate their actions with other members of a tactical air unit. To support training of fighter pilots, learning agents must be able to develop this capability, to interact and coordinate with other learning agents as well as human trainees. Still, much of the prior work within reinforcement learning for development of air combat simulations have focused on 1-vs-1 scenarios, and
often with only one learning agent. In this chapter we study how the recent state-of-the-art algorithm MADDPG [84] for deep multi-agent reinforcement learning performs on simple air combat scenarios.

In air combat, coordination could be related to many different aspects, e.g., obtaining an advantageous geometry in relation to an attacker to enable weapon delivery, efficient use of sensors to detect the enemy, or using electronic warfare systems to deny the enemy the opportunity to detect you. In this evaluation, we focus on coordinated navigation, which is the foundation for coordinated use of tactical aircraft systems.

In Chapter 3 we noted that in some training scenarios it is desirable to have synthetic agents act according to a certain nation’s doctrine for air combat. One way of achieving this is by selecting a suitable design of the agent’s reward signal and available actions. By using dense, informative reward signals the goals of the agent become clear, and its behaviour can be controlled to a higher degree than when sparse reward signals are used. By using pre-defined actions with a temporal extension, at level 3 GEN of the LACC, the agent’s behaviour is restricted to use manoeuvres that are based on intelligence information about a certain opponent’s expected behaviour.

On the other hand, Chapter 3 also informed us that advanced tactical training requires challenging opponents. If agents are given more freedom to explore and select actions freely, they can discover and exploit flaws in human-developed tactics. This can help an organisation improve their capability over time. To achieve this, abstract and sparse reward signals can be used in combination with lower level actions (e.g. at levels 1 PHY and 2 IMP of the LACC) in the design of the agent. This allows the agent to find an optimal way of achieving the higher level objectives, without being too influenced or constrained by human subject matter experts’ ideas about how to best solve the scenario under study.

Finally, Chapter 3 indicated that having mixed teams of human pilots and synthetic agents could be valuable, since large scenarios can be simulated even when there are not a sufficient number of human pilots available. However, transferring synthetic agents to mixed teams after training them in an environment populated by only synthetic agents may have a negative impact on their performance, if the interactions among human and synthetic pilots differ too much from the interactions within a team of only synthetic pilots.

The simulation scenarios studied in this chapter incorporate the aspects of agent design discussed above. We study scenarios with low-level as well as high-level action spaces, observation spaces with and without agent communication, and reward functions with dense as well as sparse feedback regarding agents’ performance. The intention is to evaluate the performance of the learning algorithm for different design choices and learning approaches, as well as to study the behaviour of trained synthetic agents when interacting with human pilots.
4.2 Method

To evaluate the potential of machine learning, in particular multi-agent reinforcement learning, as a tool for building behaviour models for synthetic pilots, we conduct a number of experiments in simplified air combat scenarios. The scenarios studied form a foundation for counter air operation, which aims to achieve air parity, air superiority, or air supremacy in relation to an opponent. The scenarios are similar to those used in the original paper of the MADDPG algorithm \cite{zhang2018multiagent}, but when implemented in our simulation engine, the increased number of time steps required to complete an episode adds additional difficulty. The learning method and simulation environment used are described below.

Learning Algorithm and Common Experiment Design

The MADDPG algorithm is used to train the agents in all scenarios. With MADDPG, the reward signal of each agent can be specified independently from the reward signals of other agents. This allows us to model that there may be conflicting interests within a team. We use a discount factor of $\gamma = 0.95$, a learning rate of $\alpha = 10^{-2}$, and train using the Adam optimizer \cite{kingma2014adam}. All simulation results are averaged over five runs with different seeds for random number generators. The policy is represented by a multi-layer perceptron, with two hidden layers, each with 64 neurons and the ReLU activation function. The agents are trained with batches of 1024 samples, sampled from a replay buffer with room for 1 million samples. We use the reference implementation of MADDPG \cite{zhang2018multiagent} in the experiments.

In the observation space of the agents, positions of aircraft and points of interest (POI) are given in a body-fixed coordinate system, while headings are given relative true north, as illustrated in Figure 4.1. All elements of the observation vector $o_t$ are normalised by their expected maximum value. In every time step, the complete observation $O_t$ of each agent, which is the input to the neural network representing the agent’s policy, is the stack of observations from the last four time steps:

$$O_t = \begin{bmatrix} o_t \\ o_{t-1} \\ o_{t-2} \\ o_{t-3} \end{bmatrix},$$

(4.1)

Simulation Environment

As simulation platform we use a Computer Generated Forces (CGF) \cite{bauer2015computer} simulation software that is part of an operational pilot training simulator.
This software is similar to a game engine, but specialised on military simulation. The CGF software simulates synthetic entities in the environment of the aircraft operated by human trainees. For our experiments we constructed reinforcement learning environments as Python modules based on the OpenAI Gym \[18\] interface, but adapted to support multi-agent learning. The OpenAI Gym interface is supported by many of the available implementations of reinforcement learning algorithms, which simplifies evaluation and comparison of different learning approaches. We call the framework GYM-CGF, and the structure of an environment is illustrated in Figure \[4.2\]. Most of the environment’s functionality is implemented in the EnvironmentCore class. This class communicates with a simulation process, running locally or on a remote computer, through the SimulationInterface, to transfer observations and actions between entities in the simulation and reinforcement learning agents that control them. The SimulationInterface is also used to load simulation scenarios in the CGF software.

Communication between the simulation and the environment module is implemented using ZeroMQ \[53\], an open-source, lightweight messaging middleware, with bindings for many programming languages, including C++ and Python. ZeroMQ makes it easy to implement several popular messaging patterns, such as request-reply, publish-subscribe and push-pull. Messages are specified using Google protocol buffers \[146\], which is a language-neutral and platform-neutral mechanism for serialisation of structured data.

To configure a specific environment, a number of delegate objects were used:
4.2. Method

Figure 4.2: Components of a GYM-CGF environment.

- **ActionDelegate**: The ActionDelegate specifies the action space of the environment (as one of the space definitions available in OpenAI Gym). During execution, it takes as input an action from this space and converts it to an ActionRequest message, which can then be sent to an entity in the simulation by the EnvironmentCore.

- **ObservationDelegate**: The ObservationDelegate specifies the observation space of the environment (as one of the space definitions available in OpenAI Gym). During execution, it takes as input a StateUpdate message from an entity in the simulation and converts it to a state observation from the observation space, which can then be presented to the agent.

- **RewardDelegate**: The RewardDelegate takes as input a state observation and calculates a reward signal, which can then be presented to the agent.

- **ScenarioDelegate**: The ScenarioDelegate manages the scenario to be simulated, including termination criteria. For each episode during training, the delegate adjusts the scenario contents as needed and generates a SimulationRequest message, which can be sent to the simulation by the EnvironmentCore.

- **RenderDelegate**: The RenderDelegate renders a view of the current state of the simulated scenario. This can be useful for debugging. We implemented a simple rendering of a map using the Python Matplotlib and Basemap libraries.
4. Learning to Interact and Coordinate

Figure 4.3: Simulation scenario for Offensive Counter Air.

4.3 Experimental Evaluation

To evaluate the performance of multi-agent reinforcement learning in air combat training we implement four scenarios with different characteristics in GYM-CGF: Offensive Counter Air, Defensive Counter Air, Defensive Counter Air with Human-Agent Teaming, and Aerial Reconnaissance. The scenarios are further described and evaluated in the following sections.

Offensive Counter Air

Offensive counter air operations aim to reduce an opponent’s capability to operate in the air, primarily by attacking high-value assets, such as air fields, aircraft, and fuel depots on the ground. In this scenario, there are three points of interest assigned as targets that should be attacked by three learning agents. The challenge of the learning agents is to allocate the three targets among themselves, and move to one target each. This mission scenario has two major phases. The first phase is the ingress, where aircraft fly a route from the home base to the target area. The second phase starts when the aircraft reach the target area, and carry out the actual attack. In our experiment, we only study the task of navigating close to the target, without any weapon delivery. We study two different configurations of the scenario in our experiments. In the first configuration of the scenario, the agents are initialised with random positions and headings in an area to the south corresponding to the home base, while the targets are initialised in random positions in a larger area to the north. The spawn areas of aircraft (in blue) and targets (in green) are shown in Figure 4.3. The displayed geographical region has a size of 100 km x 100 km. This configuration of the scenario contains both the ingress and attack phases. To further study the performance of the learn-
4.3. Experimental Evaluation

ing algorithm, and its dependence on the starting positions of agents and the design of their action spaces, we conduct an experiment with a second configuration of the scenario. In this configuration, the aircraft spawn in the same area as the targets (the large spawn area to the north), i.e., skipping the ingress phase and only requiring the agents to perform the attack phase.

The observation space of each agent is the relative position of all other agents, as well as the targets. We study two types of action spaces. The first action space is a continuous action space, which allows an agent to fly forward, or turn left or right with a load factor of 2-4 g. When experimenting with the design of the action space it was noted that using steep turns improved the exploration of the agent. The second action space is a hierarchical approach, with discrete actions that let the agent select a target and allocate it to a lower level controller, which will fly in pursuit of the target using a standard procedure. This results in a temporally extended generic action at level 3 GEN of the LACC, as discussed in Chapter 3. We terminate the action periodically every $t_{step}$ seconds, to allow the agent to evaluate the current context at level 6 FRA of the LACC and select a new action.

To promote cooperation, the learning agents receive a shared reward defined as:

$$r_t = -\sum_{i=1}^{3} \min(\|p_{t_i} - p_{a_1}\|, \|p_{t_i} - p_{a_2}\|, \|p_{t_i} - p_{a_3}\|)$$  \hspace{1cm} (5)

in each time step of the simulation, where $p_{t_i}$ refers to the position of target $i$ and $p_{a_k}$ refers to the position of attacker $k$. To carry out the task efficiently, the agents must learn to split up and attack one target each.

We train the agents for 60,000 episodes. For the low-level action space, episodes last for 500 time steps lasting 1 s each, while for the high-level action space, episodes last for 50 time steps lasting $t_{step} = 10$ s each. Agents that are hard-coded to always attack the same target are used as baselines, and their scores averaged over 1000 simulated episodes are also presented. The hard-coded baseline is strong, but not optimal, since it does not consider the initial positions and headings of agents. To perform comparatively well, the learning agents must learn to coordinate their actions based on only observations.

The training progress for the Offensive Counter Air scenario is presented in Figure 4.4, along with the score achieved by the baselines with hard-coded target allocation among the agents of the team. To make it possible to compare the results of the different settings, the sum of returns achieved by the team of learning agents ($a$) was normalised using the average return achieved by the corresponding baseline ($b$) and the average return achieved by a random policy ($r$), to get a relative score:

$$s = \frac{a - r}{b - r}.$$  \hspace{1cm} (4.2)
4. **Learning to Interact and Coordinate**

We can see that the high-level controller makes fast progress towards the score of the baseline, which it comes close to after roughly 10,000 episodes. Then there is a slight dip in performance after roughly 20,000 episodes, and the performance remains below the baseline for the rest of the experiment. While we could observe that some of the trained agents learned policies as good as the baseline, others struggle a bit in some episodes. The reason for this is possibly that in this scenario targets may spawn quite close to each other, which makes it difficult for the learning agents to coordinate based on observations alone. The low-level controller, quickly converges to a sub-optimal policy, and then does not improve during the rest of training. Possibly further training episodes could eventually lead to an improvement of the policy. When the aircraft of agents spawn in the same area as the targets, the performance is improved. Since the ingress phase is removed, and aircraft start closer to the targets, the task of learning coordination among agents is simplified.

**Defensive Counter Air**

Defensive counter air operations aim to protect territory and resources from enemy aircraft, by using a combination of surface-to-air weapons and defensive combat air patrols. In this scenario, three aircraft controlled by learning agents should escort potential threats out of their air space, in order to protect three high-value assets. Incoming threats are controlled by handcrafted behaviour models, implemented using behaviour trees \[23, 60\]. To escort a threat out of protected air space, an aircraft needs to fly within 5 km of this...
threat. The challenge for the agents controlling the aircraft is to learn to allocate threats among themselves, so that each aircraft can escort a threat out of protected air space. The scenario is illustrated in Figure 4.5. The displayed geographical region has a size of 100 km × 100 km. Before each episode of training, defending aircraft spawn in random positions and with random headings in the rectangle. Threats spawn in random positions along the arrows, and then approach along the arrows towards the high-value assets illustrated by the circles. Once a high-value asset is reached, threats will circle the asset until being escorted away by the defending agents.

The observation space of each agent is the relative position of all other agents. We study three types of action spaces. First, the same two types of action spaces used by the agents in the Offensive Counter Air scenario are studied. Then, in addition we study a third tuple action space $A = A_{\text{turn}} \times A_{\text{state}}$. Here the first action corresponds to the continuous action in the Offensive Counter Air scenario. The second action allows an agent to set its internal state as a three element, real-valued and normalised vector, which is then distributed to the other agents in the team in each time step. Previous work has shown that this type of mechanism can allow agents to develop a language for coordination of their actions [84]. In air combat simulation, the internal state of an agent could, e.g., signal its chosen priority target. This could help establish the context of decision-making for a group of agents, at level 6 FRA of the LACC.

To promote cooperation, the learning agents receive a shared reward defined as:

$$r_t = - \sum_{i=1}^{3} \min(\|p_{a_i} - p_{d_1}\|, \|p_{a_i} - p_{d_2}\|, \|p_{a_i} - p_{d_3}\|)$$
4. Learning to Interact and Coordinate

Figure 4.6: Mean and standard deviation for the training progress of Defensive Counter Air with low-level and high-level action spaces.

in each time step of the simulation, where \( p_a \) refers to the position of attacker \( i \) and \( p_d \) refers to the position of defender \( k \). To maximise the shared reward, the group needs to minimise the distance between each attacker and its closest defender.

We train the agents over 90,000 episodes when using a low-level action space, and for 30,000 episodes when using a high-level action space. For the low-level action spaces, the agents select actions at 1 s intervals, with episodes lasting for 600 time steps, while for the high-level action space they select actions at 10 s intervals, with episodes lasting for 60 time steps. As in the Offensive Counter Air scenario, we use agents that are hard-coded to always head for the same attacker as baselines.

The training progress for the high-level action space over 30,000 episodes and the training progress for the low-level action spaces over 90,000 episodes are presented in Figure 4.6, along with the score achieved by the baseline with hard-coded target allocation among the agents of the team. To make it possible to compare the results of the different settings, the sum of returns achieved by the team of learning agents was normalised using the average return achieved by the corresponding baseline and the average return achieved by a random policy, as for the Offensive Counter Air scenario.

We can see that the high-level controller quickly converges to policies that perform close to the baseline. The high-level action space automatically moves the agents towards the protected area, so that agents can start learning cooperation strategies from the start. The low-level, silent controller has an initial
plateau in performance. This is likely because the agents must first learn to move as a team towards the protected area, before being able to learn the benefits and means of cooperation. The learning progress during the second stage of learning is quite slow, and varies among different runs, as can be seen by the increase in variance. The low-level, communicating controller displays faster improvement, and also finds a policy that generates slightly more reward than the policy of silent agents. This indicates that explicit communication mechanisms can be valuable for efficient cooperation in air combat scenarios. One characteristic of this scenario that makes it simpler than the Offensive Counter Air scenario is that, by the design of the scenario, the targets that should be allocated among the learning agents are spread out geographically in each episode.

**Defensive Counter Air with Human-Agent Teaming**

This scenario is a variation of the Defensive Counter Air scenario, where one of the synthetic agents is replaced by a human operator. The intention is to study how the mixed team of humans and synthetic agents perform on the task, and how the behaviour of humans and agents differ. One challenge for the synthetic agents is that they have been trained in an environment without human participants. We use agents trained on Defensive Counter Air with the low-level action space without communication, and transfer them to the more compact scenario geometry illustrated in Figure 4.7. The displayed geographical region has a size of 70 km × 70 km. To reduce the time required for running the complete experiment, the length of each episode is also reduced to 120 steps. We define a criterion for mission success as intercepting all attackers before the end of an episode. Human pilots control their aircraft with
4. Learning to Interact and Coordinate

Figure 4.8: Illustration of the monitor setup that provides observations for human pilots in experiments with human-agent teaming. **Left:** Out-the-window view. **Right:** HDD map with view of the scenario.

We run the scenario for five iterations with each pilot. After completing the runs, pilots are given the chance to express their thoughts about various aspects of the scenario, e.g., the perceived difficulty level of the task, the performance of the synthetic agents, and the value of different forms of interaction with the synthetic agents to support coordination.

To the left of Figure 4.9 we present the mean and standard deviation for rewards received by teams with only agents compared to the rewards received by teams with a mix of agents and humans. As in the previous scenarios, we calculate a normalised score by using agents with random policies and agents with fixed target allocations. In this scenario, the learning agents outperform the baseline (with scores above one). The reason is probably that in the compact environment, the initial states of agents are of greater importance for the target allocation. We can see that the performance of the two categories of teams is similar, although teams with a human participant perform slightly better on average. From a qualitative point of view, when observing the outcome of each iteration, it could be seen that both teams were trying to split up and have one pilot approach each threat, which is the optimal tactic for the scenario. However, humans were better at quickly resolving conflicts, when two pilots started approaching the same target. We found that mixed teams of humans and synthetic agents, as well as teams populated by only synthetic agents had a success rate of 80% over the complete experiment, i.e. all threats were intercepted and escorted away in 80% of the simulations.
In discussions, humans expressed that in three of the five iterations it was clear how to allocate the threats among the pilots, while in two iterations it was not obvious what the optimal target allocation would be. The behaviour of agents received fair scores by the human pilots. The major complaint was that when there was a conflict in target allocation, with a human pilot and an agent approaching the same target, agents might not immediately realise this and select a new target. In the experiments with pure agent teams, it was noted that when conflicts arise agents may also have difficulties determining which agent has the most favourable position to keep pursuing the target.

Human pilots felt that the situation awareness map was valuable for coordinating with the synthetic pilots, but also reasoned that additional information presented in either text or speech messages, e.g. the targets selected for pursuit by synthetic pilots, would further simplify the task. This is most likely true for the synthetic pilots as well. To incorporate such functionality, the action and observation spaces of the learning agents could be modified. For instance, by letting each agent act by selecting which threat to engage, rather than acting by commanding the desired turn rate, information about the agent’s selected target becomes available, and can be distributed over data link. This would make the agent’s behaviour more explainable and transparent, and coordination with human pilots could be improved. In a similar way, by modifying the learning agent’s observation space, and including information about targets that have been selected for pursuit by other pilots in the scenario (as reported over data link), the decision-making task of the agent could be simplified.

To the right of Figure 4.9 we can see the distribution of the aircraft bank angle for agents and human pilots over the iterations of the experiment. When running the experiments and observing aircraft controlled by learning agents
in a 3D visualisation, it was noted that the aircraft were turning frequently, which can also be seen in the plot. Turns are often executed with bank angles close to ±60 degrees, which corresponds to 2 g turns, i.e. the lower end of the configured action space. Aircraft controlled by human pilots are more frequently flying straight, but when they turn they use steeper turns than synthetic agents. The qualitative behaviour of the synthetic agents is related to the design of their action spaces as well as the design of their reward signals. The action space in this experiment is a low-level action space, with continuous actions. This means that the agents can select to fly along many different routes, which makes it challenging for the agent to explore all possible solutions, and to find the optimal action for each state. Since we configured the action space to use steep turns, it also become challenging to continuously fly straight. Instead the synthetic agents alternate between left and right turns to fly along a straight trajectory. Furthermore, there is no component in the reward design that gives the agent an explicit incentive to avoid aggressive control of the aircraft. Modifying the agent’s action space or reward function could help make the agent’s behaviour more human-like and believable. The action space could be modified to allow agents to command less steep turns, or to provide more abstract, high-level actions. The reward function could be modified to penalise aggressive manoeuvres.

**Aerial Reconnaissance**

Aerial reconnaissance is reconnaissance carried out using aircraft, with a purpose to support a military strategy, e.g., localisation of enemy troops and equipment. In this scenario, two agents should visit three points of interest (POIs) as quickly as possible. The search area where the POIs are located has radius 20 km, and the agents spawn in random positions on its perimeter,
while POIs spawn in random positions within the perimeter. Examples of initial positions of aircraft (black stars) and POIs (green circles) in an episode are shown in Figure 4.10. The displayed geographical region has a size of 100 km × 100 km.

The observation space of each agent contains its own heading, the other agent’s position and heading, the positions of the POIs, and information about which POIs have been visited by either of the agents (indicated by 0 for visited, and 1 for not visited). Each agent has a continuous action space, which allows it to turn left or right with a load factor in the interval [2, 4] g. We study learning with two types of sparse reward signals. The first type of reward is a fully shared reward defined as:

\[ r_t = 50.0 \cdot n_{POI}(t) - 1.0, \] (4.3)

where \( n_{POI}(t) \) is the number of POIs visited (within 1 km) by the agents in the time step. The second type of reward is a mixed reward defined for each agent \( i \) as:

\[ r_{t,i} = 50.0 \cdot n_{POI}(t, i) - 1.0, \] (4.4)

where \( n_{POI}(t, i) \) is the number of POIs visited (within 1 km) by agent \( i \) in the time step. If both agents would happen to arrive at the same POI in the same time step, the reward is given to one of the agents by random. The scale of the reward components was selected so that they would have similar contributions to the expected return.

The rewards for visiting individual POIs can be viewed as sparse positive feedback for completing sub-goals. The shared negative reward of \(-1.0\) in each time step is intended to motivate the agents to visit all POIs and complete the episode as quickly as possible. The setting with a mixed reward still has an incitement for coordination, since due to the identical performance of aircraft, the random starting positions of aircraft, and the random positions of POIs, even competing agents are encouraged to not go for the same POI, but instead finish the episode as quickly as possible. The most useful joint policy on average should be to let the agent with the most favourable starting position visit the most POIs in each episode. To solve the problem efficiently, the agents must learn to coordinate their actions. The sparse positive reward in combination with the size of the search area makes learning challenging. The low-level action space provides an additional challenge.

By using sparse rewards in combination with a low-level action space, the risk of introducing a bias in the agents’ policies is decreased, and the possibility of finding novel behaviours is increased. For instance, learning agents could find novel ways of dealing with threats encountered during reconnaissance missions. This is not required in the scenario studied here, but finding a solution is still challenging. To help the agents learn in spite of the sparse
4. Learning to Interact and Coordinate

rewards, low-level action space, and large search area, we use curriculum learning. As discussed in Chapter 2, curriculum learning aims to support learning by successively increasing the challenge of the learning task \([11]\). One example of curriculum learning in goal-oriented environments, i.e. environments where it is desired to reach a certain goal state, is to use start states increasingly far from the goal state \([32]\). Inspired by this approach, we construct learning curricula where the agents are exposed to search areas with increasingly large radius. In our setting, the goal state is a state where all POIs have been visited, and the distance to the goal state from the starting states of the agents is measured in the number of environment steps required to visit all POIs, which is determined by the size of the search area.

Before each episode of training, the radius of the search area is sampled from a uniform distribution over the interval \(r_i = [r_{\text{min}}, r_{\text{max}}]\). We study four different learning curricula. The first three curricula have fixed, open-loop policies for adapting the challenge of the environment, while the fourth curriculum uses feedback about the learning agents' performance to decide if the challenge should be increased or reduced. Each of the fixed curricula updates \(r_i\) three times during training, according to the number of completed episodes specified in \(\text{cur}_{\text{switch}} = [n_{\text{ep}}, n_{\text{ep}}, n_{\text{ep}}]\), with values according to Table 4.1. With these fixed learning curricula we intend to investigate how the fraction of small and large search areas used during training affects the final performance of the agents. Compared to Curriculum 1, Curriculum 2 trains for a longer time on medium-sized search areas. In contrast to the other two curricula, Curriculum 3 keeps the minimum radius of the search area at 5 km throughout training. Curriculum 4 sets \(r_{\text{min}}\) and \(r_{\text{max}}\) according to the step \(s\) of the curriculum:

\[
r_{\text{min}}(s) = \max\{s - 5.0, 5.0\}.
\]

\[
r_{\text{max}}(s) = \min\{s + 5.0, 20.0\},
\]

\[
s \in \{0, \ldots, 20\}.
\]

This means that at the first step in the curriculum, the interval for the radius of the search area is \([5, 5]\), i.e., the radius is fixed at 5 km in the initial stages of learning. At the final step of the curriculum, the interval for the radius of the search area is \([15, 20]\), i.e., the radius is close to that of the target problem in each episode. The success rate of the agents is evaluated every 1000 episodes. If the agents have visited all POIs in more than a fraction \(s_{\text{threshold}}\) of the episodes, the curriculum step is increased by 1 to provide more challenge to the agents, otherwise the curriculum step is decreased by 1 to simplify the task. We set \(s_{\text{threshold}} = 0.9\), since the environment is designed so that the agents should be able to visit all POIs in each episode if an optimal joint policy is learned.
Table 4.1: Fixed, open-loop curricula used in the Aerial Reconnaissance scenario. The Switches lists specify at which time steps each Search Radius Interval \((r_0-r_3)\) is activated within a curriculum. The radius intervals are given in kilometres.

<table>
<thead>
<tr>
<th>Curriculum</th>
<th>Switches</th>
<th>(r_0)</th>
<th>(r_1)</th>
<th>(r_2)</th>
<th>(r_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cur. 1</td>
<td>([0,10k,20k,30k])</td>
<td>([5,5])</td>
<td>([5,10])</td>
<td>([10,15])</td>
<td>([10,20])</td>
</tr>
<tr>
<td>Cur. 2</td>
<td>([0,10k,30k,45k])</td>
<td>([5,5])</td>
<td>([5,10])</td>
<td>([10,15])</td>
<td>([10,20])</td>
</tr>
<tr>
<td>Cur. 3</td>
<td>([0,10k,25k,40k])</td>
<td>([5,5])</td>
<td>([5,10])</td>
<td>([5,15])</td>
<td>([5,20])</td>
</tr>
</tbody>
</table>

We train the agents for 50,000 episodes. Episodes end if all POIs have been visited within a range of 1 km, or if a maximum of 300 steps (lasting 1 s each) have been completed. We evaluate the approach by studying the trained agents’ performance for the four different learning curricula in 10,000 simulations on the goal task, i.e. the search area with radius 20 km. For comparison, we also train agents without learning curricula, on environments with radii of 5 km, 10 km, and 20 km, to see how learning performance is affected by the size of the search area. After training, we also evaluate these agents’ performance in 10,000 simulations on the goal task, to see how well policies learned on smaller search areas generalise to larger search areas.

The training progress of agents with and without curriculum learning is shown in Figure 4.11, as mean and standard deviation of the achieved return of both agents in a team over five iterations with different random seeds. For the agents learning with an adaptive curriculum, the average curriculum step is also presented. Note that the relative performance of agents learning with different reward signals or start distributions in general cannot be determined from these plots. For the same performance in terms of time required to visit all POIs, agents with a shared reward signal would accumulate higher returns, since each agent is rewarded for each POI visited. The main purpose of the plots is to study the stability of learning for different reward signals and start distributions.

We can see that open-loop curriculum learning agents with shared rewards have unstable performance when the curriculum switches to larger search areas, indicating that the agents have not learned to understand the environment and its reward signals. Performance is especially poor for curricula 1 and 2, which move faster to challenging environment configurations than curriculum 3. For curriculum 3 the learning is more stable, and plateaus after the last curriculum switch. Open-loop curriculum learning agents with mixed rewards have more stable performance as they progress through the different curricula, improving after an initial loss in performance after the initial curriculum switches, and reaching a plateau after the final curriculum switch.
4. **Learning to Interact and Coordinate**

On average, for curricula 1 and 2 learning with a mixed reward results in a higher return than the one achieved when learning with a shared reward, even though the shared reward gives each agent a reward for each POI visited. With the shared reward, agents may fail to learn the benefits of avoiding to approach a POI that another agent is approaching, since it will be rewarded
4.3. Experimental Evaluation

even when the other agent reaches the POI first. This might lead to a worse shared tactic than in the mixed setting, where the individual reward for visiting POIs will provide an incentive to agents to avoid heading for POIs that another agent is more likely to reach first.

For agents learning with an adaptive curriculum with feedback about the agents’ performance in combination with a fully shared reward, the curriculum steps forward after about 5000 episodes, but then steps back to the initial step again after 10,000 episodes, as the performance of the agents deteriorates and they fail to sustain a 90% success rate. Then performance improves again, and the curriculum step keeps increasing towards the end of learning, but only reaches around 7.5 on average, and with high variance. We could see that several iterations struggled to achieve the target success rate of 90% even for smaller search areas. Unlike the results for curricula 1 and 2, the average achieved return remains fairly stable as the challenge of the task increases, due to the fact that it increases slower. For the agents learning with a mixed reward, performance is significantly better. The curriculum reaches the final step before 40,000 episodes of learning, but then some instability can be observed, indicating that the agents have not yet learned to consistently succeed in achieving 90% success rate.

We can see that agents learning without a curriculum in the small search area with radius 5 km improve quickly, when learning with shared as well as mixed reward signals. Agents learning with a shared reward plateau after roughly 45,000 episodes, while agents learning with a mixed reward stabilise after about 40,000 episodes. There is more variance over iterations for agents learning with a shared reward. Agents learning without a curriculum in the medium-sized search area with radius 10 km also improve their performance, but display high variance, especially when learning with a mixed reward. This is likely due to the randomness of the exploration process. When learning with a shared reward, agents quickly converge to sub-optimal policies, while agents learning with a mixed reward keep improving their performance towards the end of learning. With longer training time it is possible that they could have improved further. Finally, the agents learning without a curriculum on the goal task (a search area with radius 20 km) do not display any significant improvement in performance during training.

The results of the benchmark simulations for 10,000 time steps are shown in Figure 4.12, in terms of success rate (visiting all POIs within an episode) and the average number of POIs visited per episode. For agents learning with shared reward, only learning with curricula 3 and 4 achieves higher than 50% success rate. Performance is worse for curricula 1 and 2, which proceed faster to more challenging start distributions. The agents learning on a search area with fixed radius of 5 km outperform agents learning with curriculum 1, while agents learning on a search area with fixed radius of 10 km perform very poorly, and those trained on a search area with fixed radius of 20 km have close to 0% success rate. For agents learning with mixed rewards, improved success
4. LEARNING TO INTERACT AND COORDINATE

Figure 4.12: Performance metrics on the goal task (search area with 20 km radius) of Aerial Reconnaissance for agents learning with curricula 1-4 and agents learning on search areas with fixed radius of 5 km, 10 km, or 20 km. Left: Success rate of agents learning with shared and mixed rewards, i.e. ratio of episodes where all POIs are visited. Right: Average number of POI visits per episode for agents learning with shared and mixed rewards.

rates can be observed. Curriculum learning agents have the best performance, and agents learning with curriculum 1 achieve over 90% success rate followed by curricula 2, 3, and 4 with decreasing performance, but still above 80% success rate. Agents learning on fixed size search areas perform significantly worse, but outperform their counterparts with shared rewards, except for the agents learning on a search area with fixed radius of 20 km. It can be noted that for the open-loop curricula, the order of the best curricula are reversed for agents learning with shared rewards compared to agents learning with mixed rewards. Agents learning with shared rewards can not handle the progress in challenge provided by curricula 1 and 2, resulting in a collapse of the policies, as seen in Figure 4.11. We draw the conclusion that for this environment the shared reward results in slow learning, which makes the fixed curricula unstable. Curriculum 4 avoids collapse of the policy by only increasing the challenge of the environment when the agents show strong performance, but since agents fail to achieve the required 90% success rate for larger search areas they primarily get experience from small search areas, which negatively affects the performance on the target task. Curriculum 3 manages to stabilise learning by continuing to train on smaller search area with some probability, while also training on larger search areas.

The performance in terms of average POI visits per episode resemble those observed for the success rate, since a low success rate also results in fewer POI visits per episode on average. An exception is agents learning with mixed rewards on fixed sized search areas with radii 5 km and 10 km. Here agents learning on the larger search area visit more POIs on average, although the agents learning on the smaller search area achieved higher success rate. How-
ever the difference is small and the variance is high. Overall, the agents learning with a mixed reward signal perform better than those learning with a shared reward, except when learning on a search area with fixed radius of 20 km. The best results are observed for curriculum learning agents with mixed rewards.

4.4 Related Work

The central critic of MADDPG stabilises learning in multi-agent systems compared to independent learning with single-agent algorithms. This makes learning easier, and more computationally efficient. Therefore, the introduction of MADDPG (and other approaches based on the principle of centralised learning and decentralised execution) has led to increased interest in multi-agent learning, and several works related to air combat simulation have used the algorithm since its introduction. Some works focus on developing specific capabilities, e.g. flight manoeuvres [156], decision-making for electronic warfare [53], or communication strategies in contested environments [170]. Other works focus on improving the efficiency of learning in complex multi-agent environments, which is more related to the work presented in this chapter.

Zhang et al. used a two step learning scheme to improve performance when learning with MADDPG in a competitive setting [167]. Agents were first trained in an environment were their opponents were stationary. The agents were then transferred to a new environment, with moving opponents, for further training and improved performance. In addition to the transfer learning approach, the work also studied self-play (i.e. agents training against copies of themselves) and incorporation of a pre-defined rule-base that selected actions in some states, to avoid having to learn the whole policy. Experiments demonstrated that these enhancements of the learning scheme improved performance.

Wang et al. combined MADDPG with temporally extended actions, to get a hierarchical decision-making model [150]. Instead of using pre-defined actions, as we did in this chapter, they used a pre-training phase were each lower level action was learned. In addition to the hierarchical structure of the policy, action masking was used to block selection of invalid actions, e.g. trying to chase a target when no target has been detected. This can make learning more efficient.

Zhao et al. used meta-learning to improve the generalisation of learning for collaborative swarms of UAVs [171]. Meta-learning is a technique that can learn meta-models in one environment, and then use them to improve learning performance in another environment [53, 11]. Model Agnostic Meta Learning (MAML [31]) was combined with MADDPG, and evaluated in experiments in environments with variations between episodes of learning, resulting in a dynamic battlefield environment.
4. Learning to Interact and Coordinate

4.5 Summary

In this chapter, we conducted an experimental evaluation of multi-agent reinforcement in four simplified scenarios related to counter air operations: Offensive Counter Air, Defensive Counter Air, Defensive Counter Air with Human-Agent Teaming, and Aerial Reconnaissance. Learning agents’ capability to perform in these scenarios is relevant whether they act as friendly or hostile entities in the scenarios.

In the Offensive and Defensive Counter Air scenarios we saw that actions with a temporal extension can significantly improve learning performance. In addition, doctrinal behaviour could be encoded in such actions, to make the behaviour of synthetic agents appear more realistic. The results in Defensive Counter Air with Human-Agent Teaming illustrated that learning with low-level actions can instead result in unwanted qualitative effects, such as frequent turning of the aircraft, even when the quantitative results indicate decent behaviour. Such unwanted side-effects could be reduced by adding additional objectives in the reward function, e.g., penalties for unwanted behaviour. However, that would make the reward design more complicated. For instance, flight manoeuvres that are natural in close range combat may look unnatural if executed while in cruise without any enemies in sight.

The results in the Defensive Counter Air scenario showed that communication mechanisms can improve learning performance, and as discussed, such mechanisms could to some extent be implemented in human-agent teaming as well.

The results of the Offensive as well as Defensive Counter Air scenarios, showed that when learning with low-level actions, agents may need to learn policies for reaching sub-goals before a policy for completing the full mission can be found. To make learning more efficient, the behaviour of the agent could be made dependent on the current context of the scenario. For instance, different policies could be used to handle the ingress, attack, and egress phases of an offensive counter air scenario. The full policy could be organised as a decision-making hierarchy, where an agent at the top of the hierarchy is responsible for framing the current situation at level 6 FRA of the LACC, and then distributing the context to lower level agents responsible for, e.g., navigation, sensor management, and weapon delivery.

The results in the Aerial Reconnaissance scenario showed that the reward design can significantly affect the performance of learning agents even for simple problems, and that by constructing a learning curriculum, learning performance can be significantly improved. However, constructing such curricula for more complicated problems can be challenging. Instead, a learning approach could be used for defining the curriculum as well, in a similar way that human instructors would use their acquired knowledge about the domain to update the training contents for a human trainee. This would require the agent responsible for the curriculum to assess learning agents’ current perfor-
mance, and determine how to adapt the environment to achieve a suitable change in difficulty.

In this study, we used the centralised training and decentralised execution (CTDE) approach. Since the learning agents in our experimental setup have no interactions with humans during training, there is a risk that the learned policies will overfit to the behaviour of the other synthetic, learning agents in the environment. This could result in a loss of performance when the agents need to interact with humans, who may act differently than synthetic agents. In addition, it is expected that behaviour will vary among different human pilots, especially when pilots are in training and perfecting their skills. One way to avoid the risk of overfitting is to enforce diversity in the population of agents, e.g. by learning ensembles of policies, as suggested by [84]. Another way could be to equip the agents with a capability to quickly adapt their behaviour after being transferred from the development environment to an operational system.

We also note that the reference implementation of MADDPG that we used does not implement prioritised experience replay, since experiences from all agents need to be from the same time step when training the central critic. This may have a negative impact on the performance when learning with a large replay buffer, since the experiences most valuable for learning may not be sampled very often.
In this chapter, an architecture for a simulation-based pilot training system is presented. The architecture automates parts of training adaptation and training delivery by incorporating learning agents as system elements. Based on the proposed architecture, the findings of our user study, and the results of our practical experiments, we then discuss design approaches and solution concepts that could produce agents with desirable capabilities and characteristics for pilot training. This chapter is based on Paper IV \[66\] and Paper VI \[67\].

### 5.1 Architecture for Training Systems using Learning Agents

A system architecture for a simulation-based pilot training system, which incorporates learning agents for improved efficiency and effectiveness, is illustrated in Figure 5.1. This chapter discusses how to realise the system elements of the architecture, based on the findings of our user study and practical experiments, as presented in Chapter 3 and Chapter 4.

The architecture integrates agents to support organisations and instructors in adapting training to trainees’ training needs (Scenario Adaptation Agent), and delivering it in an efficient manner (Synthetic Trainer Agent). In training scenarios, Synthetic Trainer Agents participate as synthetic role-players, with the same purpose as human role-players. These agents act in one of the roles
of the scenario, either as opponents or teammates of the trainee pilots. Their major goal is to provide a stimulating training environment to the trainees. Offline, between training sessions, the Scenario Adaptation Agent takes the role of an instructor and analyses data generated in past sessions to identify trainees’ weaknesses and strengths, and then adapts training session contents to maximise future improvement of proficiency.

The goals of the Synthetic Trainer Agent are modelled through its Reward System, which captures important features for successful decision-making in air combat scenarios. An individual agent’s preferences among reward features are determined by the agent’s Utility Space and Utility Function. The Synthetic Trainer Agent’s ability to perceive the state of its environment is essential for decision-making in the complex domain of air combat. The agent’s perception is formed by two components: the agent’s interface to the fused information of the aircraft’s sensors (Observation Space) in combination with the agent’s internal beliefs regarding the state of the world, based on its past observations (World Model). The Decision System of the Synthetic Trainer Agent realises its capability to learn how to act (using the actions of its Action Space) based on past experience, as well as its capability to evaluate and plan future actions based on its learned understanding of the dynamics of its environment, e.g., its ability to predict future behaviour of other agents. Agents acting as synthetic role-players enter the training environment through computer generated forces software. Agents are first trained offline, and then interact with human trainees in training sessions, which can potentially provide data that support further adaptation of the agents’ decision-making policies.

Just like the Synthetic Trainer Agent, the Scenario Adaptation Agent learns its policy through reinforcement learning. The state considered when making decisions consists of inferred User Needs to meet Training Goals, based on User Profiling. The actions of the agent are related to Adaptation of
5.1. Architecture for Training Systems using Learning Agents

the contents of training sessions, and the rewards received are related to the changes in Trainee Performance over time.

We provide further discussions regarding these components, and ways of realising them, in the sections below. When appropriate, we make references to the discussion about levels of autonomy in cognitive control in Chapter 3, as well as the results of the practical experiments in Chapter 4, to illustrate how the elements of the architecture contributes towards modelling fighter pilot behaviour and providing high-quality training.

For review, levels 1 PHY and 2 IMP are related to HOW control is realised by a system. Levels 3 GEN and 4 VAL are related to WHAT is done in terms of generic plans and value-based decision-making. Levels 5 EFF and 6 FRA are related to WHY the system exists, through framing of the current situation and effect-based decision-making.

**Reward System, Utility Space and Utility Function**

For reinforcement learning agents, the goals that should be achieved are expressed through a reward signal. In the proposed architecture, the reward signal is generated by each agent’s internal reward system, based on current and past states of the world, according to the agent’s perception. In a standard Markov decision process (MDP) the reward signal is a scalar. To model the conflicting objectives of air combat training, we instead use a Multi-Objective MDP (MOMDP) 114, which provides vector-valued rewards. The values of the reward vector are based on important features of the scenario, which should affect the agent’s decision-making over time. For instance, rewards could be given for achieving advantageous geometry relative opponents, for detecting other entities while avoiding being detected, for missile hits, and for sensible resource management. This information supports control at level 4 VAL of the LACC. The overall value of states and actions are determined by applying the agent’s utility function $U$ over the vector returns:

$$u = U(V^x(s))$$

(5.1)

This gives a scalar value $u$ that supports ordering of policies.

The reward system design is one option for infusing domain knowledge in the behaviour of the learning agent, to bias the learning process towards desirable characteristics, e.g., making an agent learn behaviour which is in line with a certain doctrine. By using different combinations of reward components and utility functions, a diverse set of agents can be created, which can make training more varied and stimulating. The components of the reward vector represent the objectives of the agent, and finding optimal policies results in a multi-objective optimisation problem.

The utility space of the agent models which types of utility functions are available. This information can be used by the agent when searching for
5. Concept Design

policies. The information could also be used when reasoning about other agents’ behaviour. By observing another agent’s behaviour over time, it can be possible to identify which utility function is the best match for that agent’s behaviour. Having an idea about the utility functions of other agents makes it possible to predict their future behaviour.

Linear utility functions are suitable for modelling trade-offs related to monetary values, such as trade-offs between investments and return on investment in a business. In air combat simulation, they could be useful for modelling trade-offs related to combat value, e.g. modelling the utility of firing a missile at different types of targets on the opponent side. Polynomial utility functions can be used to model non-linear trade-offs between objectives, e.g. modelling that the cost of firing a missile increases in a non-linear way when the number of missiles carried by the aircraft decreases. Utility functions with thresholds can be used to model limits that must be met, e.g. limits related to a tactical unit’s probability of survival, and limits related to the probability of collateral damage during a mission. Threshold lexicographical ordering (TLO) can model that there is an ordering of objectives with constraints that must be fulfilled to a limit before other objectives may be considered. In air combat simulation, it is natural to ensure probability of survival to a certain limit before considering approaching and attacking a target.

One way to construct the reward system is to let a human pilot demonstrate how to solve a certain task, and then inferring a reward signal from this information, or simply rewarding the agent for behaving in a similar way as the human pilot. Demonstrations are often used to support training of human pilots, so this approach leads to a human-machine interface that feels natural for the instructor, and reduces the need for explicit programming of agents.

Since dense reward systems, which give frequent feedback to the agent, as well as rewards based on demonstrations, introduce a bias in the agent’s policy, they may prevent the agent from finding an optimal policy. To create very challenging opponents it may instead be desirable to use sparse reward signals, e.g., only rewarding the agents for winning a fight according to some metric. This allows agents to freely explore the world, and novel tactics and doctrines may emerge. It may also be easier for instructors to define such reward functions, since they have lower complexity than dense reward functions. Our experiments in Chapter 4 illustrated that sparse reward signals can make learning challenging. One way of improving performance is to include intrinsic motivation in the agent’s reward system, e.g. rewarding the agent for reaching novel states or for solving auxiliary sub tasks in addition for completing the full mission.

Agents that are to act as synthetic trainers for humans can not only consider components of the tactical scenario, they must also include information about the trainees’ learning needs in their reward system, so that the decisions made during a training session are based on reasoning about training
5.1. Architecture for Training Systems using Learning Agents

effect at level 5 in the LACC. This includes observing the trainees’ proficiency in different aspects of air combat, and identifying ways of giving the trainees the right stimulation to improve their proficiency over time. This process is supported by analysis of data from past training sessions offline, through the Scenario Adaptation Agent’s profiling of trainees, and inference of training needs to meet the organisation’s training goals. Agent behaviour is then further adjusted during the progression of a training session, in a similar way as human role-players would adapt their behaviour to the performance of trainees. Just as for risk-aware decision-making, decision-making for optimisation of trainee’s learning experience are expected to benefit from support for non-linear utility functions. For instance, utility functions could incorporate thresholds corresponding to sufficient proficiency achieved for each learning objective. This would allow agents to adapt their behaviour to focus learning on areas where the greatest improvements can be achieved.

Observation Space and World Model

The design of the agent’s observation space determines which features of the environment will be considered when making decisions. As illustrated in our analysis of decision-making in air combat scenarios, human pilots use low-level features (level 1 PHY of the LACC) as well as more abstract value-based information (level 4 VAL of the LACC) to support their decision-making. For efficient learning of policies, agents should be supported by similar information. This includes, e.g., knowledge about the performance of own and opponents’ vehicles, sensors, and weapon systems, which human pilots would have acquired in theoretical study. To further support synthetic agents in adapting the training environment to the needs of human trainees, they could be provided with information that is not normally available to human pilots. For instance, biosensors attached to trainees could be used to estimate their cognitive states, e.g., workload, stress, and engagement. This information could then be used to adapt the behaviour of synthetic pilots and other aspects of the environment, to improve the training effect.

To realise behaviour perceived as intelligent, agents can not act only based on the immediate observation of the world, but must instead consider its whole history of observations. This functionality is realised by the agent’s world model, which uses memory mechanisms to learn an abstract model of the state of the world, which can support decision-making. The model can be infused with domain knowledge, by explicitly modelling such features that human pilots believe are important for success in air combat, for instance, predictions regarding other agents’ goals, beliefs, and future behaviour. To support adaptation of behaviour to trainees’ current training needs, the world model should also provide abstract information related to training effect, e.g., estimates of trainees’ proficiency. The functionality of the world model enables
the agent to frame the current situation, and to reason about the effect of its actions (levels 5 and 6 of the LACC).

One challenge for learning agents is that they typically learn their policies in an environment populated exclusively by other synthetic agents, i.e., they do not interact with humans. The reason for this is the large number of iterations required for learning algorithms to converge. As identified in our user study, agents that act as team members of trainees must learn to understand trainees’ intentions, while agents that act as opponents should be able to identify and exploit flaws in human-developed tactics, which may change over time. Therefore, for agents to interact effectively with humans in training sessions, they need to have the capability of adapting their behaviour to a wide range of teammates and opponents. One way to achieve this is to maintain a diverse population of agents while learning new policies, and to assemble teams of agents by random sampling from this population before each episode of learning [61]. Since this forces each agent to learn while interacting with several different types of other agents, the agent will hopefully learn a policy that is general enough to support interaction with humans as well. Another approach is to use meta-learning, where agents learn to model characteristics and behaviour of other agents based on few observations [102, 103]. In meta-learning, an agent learns a meta-model of common properties of other agents in the environment, as well as an ability to use this model to make predictions regarding the characteristics and actions of an agent that it encounters for the first time. Provided that human behaviour is not too different from that of synthetic agents, a modelling capability learned from interactions with synthetic agents could be used for modelling of humans as well. During training sessions, such an approach could be used as a basis for modelling a specific human trainee.

### Action Space and Decision System

As observed in the practical experiments in Chapter 4, the design of the agent’s action space has great impact on its ability to explore, and will affect its final learned behaviour. For air combat simulation, it may be desirable to constrain the behaviour of the agent, so that it resembles a certain opposing force. By using masking of action spaces, actions can be made available for selection only when certain conditions are fulfilled. Such approaches have been used to make sure that learning agents abide to the rules of games [123, 148]. In air combat simulation, for instance, rules of engagement can be encoded in the action space, to restrict when and how target engagement is allowed.

The experiments with offensive counter air in Chapter 4 illustrated that scenarios with sub-tasks can have a negative impact on learning performance. In the offensive counter air scenario, agents with low-level action spaces struggled to learn ingress and target attack. By learning individual policies for each sub-task, e.g. one risk-aware policy for ingress to a target area and one policy...
for attack on targets of interest, performance can be improved. By including temporally extended actions in the design of the agent’s action space, it becomes possible to learn a policy over actions at level 3 GEN of the LACC. The options framework for hierarchical reinforcement learning provides a formalism for learning with temporal abstractions. An option $\omega \in \Omega$ is defined as a tuple $(I_\omega, \pi_\omega, \beta_\omega)$, where:

- $\Omega$ is the set of available options
- $I_\omega$ is the initiation set, specifying in which states the option can be selected
- $\pi_\omega$ is the intra-option policy, i.e., the policy used once the option has been selected
- $\beta_\omega$ is the termination condition of the option, specifying the probability of the active option terminating in a state, to allow a new option to be selected

The temporally extended actions that we used for approaching targets in Chapter 4 can be viewed as options that can be selected in any state, and that run for a fixed number of time steps before being terminated. In real-world air combat scenarios, the initiation set can be used reduce the number of actions available for selection in each time step, based on the current context at level 6 FRA of the LACC. For instance, missiles should not be fired if no targets have been detected, and evasive manoeuvres need not be performed if there are no incoming threats. Termination conditions can be used to model typical triggers for context switches, such as targets and threats detected by the aircraft’s sensor systems, or incoming orders from a command and control centre.

As discussed in the summary of Chapter 4, another way of structuring the decision-making is to form a hierarchy of agents, where each agent is responsible for one of the tasks required for success in air combat, e.g. risk-aware navigation, sensor management, and weapon delivery. These tasks need to be executed in parallel, and by allocating them to different agents the decision-making problem for each agent can be simplified. However, efficient cooperation among the agents is required for mission success. Feudal reinforcement learning can be used to create this type of hierarchies, where managers at the higher levels set tasks that are to be solved by lower level agents. In air combat simulation, feudal reinforcement learning could be combined with options. For instance, an agent responsible for navigation could use options to more efficiently learn how to navigate depending on the current context of the simulated scenario.

In addition to the performance boost observed in Chapter 4, another benefit of temporal abstractions and hierarchical reinforcement learning is that
agents’ policies can become easier to understand [124]. This is because there is a structure in the decision-making of the agent, which resembles the way humans reason about decision-making. Similar structures can also be found in the methods used for building decision-making models by hand, e.g. behaviour trees.

The options used for air combat simulation could be handcrafted, to replicate how temporally extended actions are executed according to a nation’s air combat doctrine. This is a natural approach when the designer has a clear idea about how an extended action should be performed, e.g., actions that have been optimised based on the laws of physics, such as missile guidance and evasive manoeuvres. Using handcrafted options as building blocks for learning tactics is also more likely to result in behaviour that is believable to humans, than learning with low-level actions alone, which can sometimes have undesirable effects, as illustrated in our practical experiments presented in Chapter 4. For areas where there is greater uncertainty regarding how the agent should act to solve a task, there are also algorithms that make it possible to learn hierarchical policies from scratch, e.g., the option-critic architecture 9, the double actor-critic 168, and feudal networks 147, which makes it possible to discover complex, novel forms of actions.

As noted in the discussion on reward systems and utility functions, designing air combat policies is a multi-objective optimisation problem. Multi-objective reinforcement learning (MORL) provides systematic methods for learning sets of policies that are Pareto optimal (see Chapter 2). We believe that this is a natural approach for reinforcement learning in the air combat domain, where trade-offs between conflicting objectives are often required. The method supports decision-making at level 4 VAL of the LACC. To adjust training to fit the needs of individual trainees, suitable agent policies can be selected from the set of Pareto optimal policies 73, 71, 68.

In MORL, there are two types of optimisation criteria that are used when learning policies, scalarized expected returns (SER) and expected scalarized returns (ESR):

\[
V^u_{\pi}(s) = U\left( E\left[ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right] \right), \text{SER}
\]

\[
V^\pi(s) = E\left[ U\left( \sum_{k=0}^{\infty} \gamma^k r_{t+k} \right) \mid s_t = s \right], \text{ESR}
\]

The SER criterion aims to optimise the average outcome of several episodes, while the ESR criterion tries to optimise the average outcome of each episode. For air combat training, the ESR criterion may be the most suitable one, since from an operational perspective pilots want to optimise their chances of survival in each mission, rather than their average survival rate over a complete campaign. We can see that for linear utility functions the criteria are the same, since the positions of expectation and utility functions
can be interchanged. However, as discussed in the section on utility functions, we expect that the utility functions of a synthetic fighter pilot will contain non-linear trade-offs between objectives, as well as thresholds for acceptable risk-taking.

To further adapt agents’ behaviour to fit trainees’ needs, planning algorithms can be used to adjust the agents’ policies online, while a training session is in progress. These algorithms can use the agent’s world model to do simulated rollouts, to explore the effects future actions would have on the outcome of the mission. One family of planning algorithms that has had great success in, e.g., games of various forms is monte-carlo tree search (MCTS)\footnote{MCTS can be combined with the learned value functions of the agent, to improve its performance.}. In training scenarios, planning could be used to adapt behaviour to maximise the current utility of the agent, which is related to the training effect of trainees, and level 5 EFF of the LACC.

**Training Environments**

The training environment is where agents acting as synthetic role-players interact with human trainees. Here, it is desirable to have high-fidelity models for the vehicles operated by the actors in the simulated scenario, as well as environment properties of various sorts, e.g., weather effects. However, when learning agent behaviour using current state of the art reinforcement learning techniques, many iterations of missions are required, leading to long simulation times if a complex simulator is used. For this reason, it is valuable to have the possibility to adjust the fidelity of the simulation in several steps. The initial learning can then take place in lower fidelity environments, for many iterations, and the learned policies can then be successively transferred to environments with higher fidelity models for fine-tuning.

To achieve high efficiency, the design of the scenarios used for training synthetic agents is also important. If we use sub-tasks to model missions and for structuring the decision-making system, mini-scenarios for each sub-task can be created. These mini-scenarios can be used for initial training of the lower levels of a decision-making hierarchy. Then, further training can be conducted in more complex scenarios, while at the same time training the higher levels in the decision-making hierarchy to determine contexts (at level 6 FRA of the LACC) and make effect-based and value-based decisions (at levels 5 EFF and 4 VAL of the LACC).

**Adapting Training to Inferred Training Needs**

As discussed in Chapter \footnote{As discussed in Chapter 2, an adaptive training system needs a measurement function to judge the proficiency of the trainee, an adaptive variable that can be used to modify the training environment, and an adaptive logic to modify the adaptive variable based on the measurement. In our architecture, the}, an adaptive training system needs a measurement function to judge the proficiency of the trainee, an adaptive variable that can be used to modify the training environment, and an adaptive logic to modify the adaptive variable based on the measurement. In our architecture, the
5. Concept Design

Scenario Adaptation Agent helps realise the measurement function as well as the adaptive logic. The Synthetic Trainer Agent is part of the adaptive variable, and also helps realise part of the adaptive logic.

To support adaptation of simulation contents and agent characteristics to current training needs, the Scenario Adaptation Agent should learn a model of different aspects of trainees’ proficiency, based on their performance in past training sessions. This modelling could probably be supported by similar techniques as those used to create the Synthetic Trainer Agent’s world model. Performance measurements can include, e.g., measurements describing a pilot’s flight path, risk exposure, resource management, and success rate in engagements in missions. Such evaluation metrics would ideally be captured in the form of a reward function, which could then also be used to train synthetic agents. The model of trainees proficiency is used as input to the Synthetic Trainers that participate as role-players in training sessions, and affects their utility for different types of behaviour, to achieve maximum training effect. In addition, Synthetic Trainers should adapt their behaviour throughout the training session, based on how the simulated scenario unfolds. The trainee profiles created by the Scenario Adaptation Agent could be viewed as prior probability distributions for characteristics and behaviour of trainees that are in training, which are used as a starting point by the Synthetic Trainer Agent, who then calculates posterior distributions based on observations during the execution of training sessions. This corresponds to having agents with a capability of perception and decision-making at level 5 EFF of the LACC, which is, of course, a highly challenging task. However, recent advances in agent modelling using machine learning techniques have shown promising results in predicting the beliefs, goals, and future actions of agents [103, 102].

In addition to supporting adaptation of agent behaviour, the Scenario Adaptation Agent should also be able to adjust the contents of training scenarios, so that suitable components for improving trainees’ performance are included, and so that the level of challenge is at an appropriate level for the current proficiency of the trainee. This could include formation of suitable teams of human trainees and synthetic pilots, allocation of tasks that are relevant for the training needs of the participants of each team, and modification of the environments where missions are executed, e.g. adjusting the number and placement of opponents’ forces. Here there is an overlap between training of humans and training of synthetic agents’ behaviour. For efficient learning in complex domains, synthetic agents can also be exposed to increasingly challenging problems using curriculum learning [12, 24]. As demonstrated in the practical experiments in Chapter 4, this can significantly improve the performance of agents. It is possible that curriculum learning techniques that have proven effective for training of synthetic agents could be adapted for training of humans as well, but further research on the topic is required.
5.2 Multi-Objective Learning for Training Simulations

As observed in the practical experiments in Chapter 4, constructing learning curricula by hand can be challenging. If the challenge of training tasks are set too high, learning agents may fail to perform well enough to move to the later stages of a closed-loop curriculum. If the challenge of training tasks are set too low, learning agents may perform well on the training tasks, but still fail to learn strong policies for a challenging target task. As suggested in the summary of Chapter 4, reinforcement learning could be used to control the curriculum generation, to remove the need for manual designs from scenario developers.

Recent work have proposed methods for curriculum learning that could be of interest in adaptive training systems as well. For instance, Dennis et al. proposed an approach for unsupervised environment design [27]. An adversary agent was used to learn how to adjust parameters that control various aspects of training environments, to provide increasingly challenging tasks to a learning agent. To ensure that the generated environments were solvable, an antagonist agent allied with the adversary was introduced in the environment. The adversary was then trained to maximise a regret measure defined as the difference in performance between the learning agent and its antagonist. The method was extended by Gur et al. to support generation of environments with sub-tasks [41], which are often present in air combat scenarios, as discussed in previous sections. A difficulty measure was included in the reward function of the environment generator’s policy, to avoid generating environments that are not possible to solve.

5.2 Multi-Objective Learning for Training Simulations

Prior work on reinforcement learning in the domain of simulation-based training has focused on single-objective learning, with scalar rewards. However, our domain analysis indicates that producing contents for simulation-based training is a multi-objective decision-making problem. This section presents a summary motivation for using a multi-objective, utility-based approach to reinforcement learning in the domain of training simulation. Different motivating scenarios for application of multi-objective reinforcement learning proposed by Hayes et al. [45] are presented in Figure 5.2. We discuss these scenarios and their applicability when constructing agents for simulation-based air combat training below.

In the unknown utility function scenario, the user’s utility functions is not known a priori. To prepare for possible utility functions that may be encountered in the future, a solution set is calculated. Once the utility function is known, a policy that maximises the corresponding utility can be chosen and executed. In adaptive training, the exact steps of trainees’ training curricula cannot be expected to be known a priori. Instead, having the solution
5. Concept Design

Figure 5.2: Motivating scenarios for a multi-objective approach to reinforcement learning [45]: (a) the unknown utility function scenario, (b) the decision support scenario, (c) the known utility function scenario, (d) the interactive decision support scenario, (e) the dynamic utility function scenario, and (f) the review and adjust scenario.

set makes it possible to quickly adapt the challenge of the simulation as the proficiency levels of trainees change.

In the decision support scenario, the user is not able to fully express the utility functions. This situation is common-place in simulation-based training, where the instructors are domain experts, in for instance air combat, but may have little or no expertise in formulating utility functions for learning agents. As in the unknown utility function scenario, instead of requiring that the user formulates a utility function for learning a single solution, a set of solutions
5.2. Multi-Objective Learning for Training Simulations

can be calculated. This set can then be presented to the user to support selection of the most appropriate policy. For instance, by running simulations with agents that act according to different utility functions, their behaviour and important metrics could be presented for different parts of the simulated scenario.

In the known utility function scenario, the user’s utility function is known, but a multi-objective approach can make it easier to solve the decision-making problem. For instance, in air combat simulation utility functions are expected to be non-linear in many settings, e.g., in risk-aware decision-making the utility function may contain thresholds for various forms of risky behaviour. Many popular single-objective reinforcement learning algorithms do not support non-linear utility functions, since they rely on an assumption of additive returns. Therefore, to learn with non-linear utility functions in single-objective settings a transformation of the MDP may be required, which can result in a loss of performance.

In the interactive decision support scenario, the initial setting is the same as in the decision support scenario. However, instead of using a learning phase followed by a selection phase, a single phase involving interactions with the user is used to find a suitable single solution. The interactions with the user help guide the learning of the agent towards the relevant parts of the value space, making learning more efficient than when learning a full coverage set. As our domain analysis illustrated, even simplified air combat scenarios can require many time steps, and achieving efficient learning is important to enable quick adaptation to user preferences, for instance when new aircraft systems are introduced in the operational environment.

In the dynamic utility function scenario, the utility function of the user changes over time. In this scenario, we want to adapt as efficiently as possible to the new setting as changes in utility occur. In adaptive training, this scenario corresponds to adapting the simulation to provide the highest utility of training to the trainee at any time. Changes in the utility function can be the result of changes in the trainees’ proficiency, or caused by changes in the operational environment. By searching for new solutions only when changes in user utility are detected, the total computational cost of learning can be reduced compared to learning a full coverage set, although at the cost of a delay in adaptation.

In the review and adjust scenario, there is uncertainty about the user’s utility function as well as the problem formulation. By learning a solution set and presenting it to the user, support is provided for selecting a single solution that seems to be the most promising one. In the execution phase, the effects of the solution are reviewed by the user. Depending on the outcome of the review, based on the new information a new solution set may need to be learned (e.g. after adding a new objective to the MOMDP) and/or the selection phase may need to be repeated.
5. Concept Design

In addition to the motivating scenarios above, as pointed out by Vamplew et al. [143], there are additional benefits of using vector rewards for reinforcement learning. One aspect that is of high importance in simulation-based training is the reward signal’s effect on the transparency of the agent’s decision-making. As identified in Chapter 3, users feel that it is important to be able to explain the behaviour of synthetic agents in debriefing sessions. If scalar rewards and scalar value functions are used, it is possible to identify that, in a given state, action A was chosen instead of action B since action A had an expected return of 53, while action B only had an expected return of 45. This is not very informative for users who do not know the details of the underlying agent design. If we instead use vector rewards and vector value functions, we have more information available for analysis of agents’ behaviour. For instance, in a training session it could be observed that two synthetic agents entered a state of high risk. One of the agents decided to retreat, while the other agent decided to continue its approach towards the enemy’s air space. When inspecting the two agents’ utility functions, it might turn out that the retreating agent had a lower priority for achieving tactical mission goals in relation to its priority for risk management.

The vector value functions learned by synthetic agents can not only be used to support debriefing of training sessions. They could also be used to support development of new air combat tactics. The added information compared to scalar rewards makes it easier to transfer the policies learned by the synthetic pilots to human decision-makers.

5.3 Summary

In this chapter, we presented a concept for a user-adaptive training system, which helps instructors provide training that fits the competency of individual trainees. The concept is based on knowledge about the domain of air combat training and its users, gained through interviews, survey responses, and practical experiments. The system architecture has two major elements, which enable training adaptation: the Scenario Adaptation Agent and the Synthetic Trainer Agent.

The Scenario Adaptation Agent is responsible for the high-level adaptation, acting in a similar way as an instructor. This includes a profiling of the trainee based on past performance, and an identification of the current training needs. To meet these needs, the training environment is adapted through changes in training scenarios, training missions, and tasks allocated to session participants.

The Synthetic Trainer Agent participates as an actor in training scenarios, acting in a similar way as a human role-player. Its behavioural characteristics are adjusted through its utility function by the Scenario Adaptation Agent,
in a similar way as human role-players can adapt their behaviour based on input from an instructor.

We discussed requirements related to the various components of the architecture, as well as different approaches to realise them. When constructing these agents, we propose to follow the utility-based approach to multi-objective reinforcement learning, since its use cases match those identified in our domain analysis and user study. As a first step, Part II of the dissertation presents and evaluates two realisations of the Synthetic Trainer Agent.
Part II

Utility-Based Reinforcement Learning for Simulation
In our domain analysis and concept design we identified the need for being able to adapt agent behaviour over time, to be able to provide user-adapted training. Since training instructors and other subject matter experts can not be expected to have expertise in constructing behaviour models for synthetic actors in agent-based simulation, it is important to develop methods and tools that can support users in finding suitable policies for their simulation scenarios. As discussed in Section 5.2, the decision support and interactive decision support scenarios of multi-objective reinforcement learning deal with this problem by learning sets of Pareto optimal policies, which can be demonstrated to the users to support their selection of the most suitable agent behaviour. This chapter studies how to efficiently learn such policy sets by representing them with a single neural network. This chapter is based on Paper I [73], Paper II [71], and Paper V [68].

6.1 Providing Decision Support to Instructors

In our concept design, presented in Chapter 5, we identified that it would be useful to have an interface that allowed instructors to adapt the behaviour of synthetic agents by specifying their relative preferences for a set of objectives, e.g. tactical mission goals, resource consumption, and safety. This would make it possible to easily adjust agent behaviour between training sessions, so that instructors become less dependent on simulator engineers for making such changes. By adjusting agent behaviour to the needs of the participants of
each training session, a more stimulating and effective training environment can be provided. For instance, a suitable level of challenge can be provided to novice trainees in basic training, or less predictable synthetic opponents can be provided in advanced mission training for experienced pilots. When using multi-objective reinforcement learning to create agent behaviour models, changing the agent’s behaviour corresponds to changing its utility function.

Specifying which exact utility function to use for each agent may be challenging for the user. Instead, the decision support and interactive decision support scenarios of multi-objective reinforcement learning propose to learn a set of policies, which can be presented to the user to support the selection. By learning an approximate convex coverage set (CCS) [45], it becomes possible to provide decision support to the users by allowing them to study agent behaviour for different utility functions. In simulation-based training, the policies of the CCS are building blocks that can help create training scenarios with diverse dynamics in terms of the interactions between human trainees and synthetic actors. Having the CCS also means that the simulation can be adjusted quickly if user needs change, since no additional training of agents is required. However, learning multiple policies comes with an increased computational cost compared to learning a single policy for an agent with static behaviour.

This chapter addresses the need for efficiently providing decision support, adaptiveness, and diversity in agent-based simulation used for training. Our goal is to train agents whose behaviour can be tuned at runtime to fit the current training needs of a trainee using a simulation-based training system. To achieve this goal, we propose an architecture for a tunable actor (T-Actor) that allows us to adapt existing reinforcement learning algorithms to multi-objective environments and multi-objective reinforcement learning. T-Actor provides decision support to instructors by learning a set of Pareto optimal policies, represented by a single neural network conditioned on the agent’s preferences over objectives. The objectives can be related to interactions with the environment or other agents in the system. The conditioning of the policy means that the agent’s behaviour is not static, instead it can be adjusted between training sessions to fit each session’s training goals. We empirically study the performance of the tunable actor, and show that it has the ability to approximate the policies of several agents trained with a diverse set of fixed objective preferences. This shows that T-Actor makes it possible for subject matter experts to tune agent behaviour for each training sessions, to adjust the dynamics of the simulation. In addition, our approach results in more efficient learning compared to training every policy from scratch.
6.2 Related Work

As a result of the challenges related to constructing high-quality synthetic agents, there are many examples of machine learning, including some examples of multi-objective learning, being used for modelling behaviour in agent-based simulation and related domains. For instance, Bone et al. used reinforcement learning to model how stakeholders with different objectives affect land use change [17], and Rogers et al. used multi-objective optimisation based on genetic algorithms to calibrate the parameters of an agent-based model of a financial market [112]. While these works focus on analysis, the work presented in this chapter focuses on interactive simulations for training.

Within the domain of Serious Games [78], Sawyer et al. used multi-objective reinforcement learning to support construction of tutorial planners [116]. A tutorial planner is used to adapt the gameplay of lessons to fit an individual student, to improve learning efficiency. They considered the problem of balancing two objectives: student learning and student engagement. They first built a model based on logs from student interactions with a learning environment, and then used Convex Hull Value Iteration [10] to learn a tutorial planner for educating middle school students in microbiology. This approach is similar to the Scenario Adaptation Agent in the proposed architecture for an adaptive training system, presented in Chapter 5. This chapter focuses on the Synthetic Trainer Agent, and a method for constructing agents that can be used as adaptive variables when adjusting simulation contents to fit trainees training needs.

Machine learning techniques have also long been used to build game-playing agents. For instance, supervised learning and recorded data from Starcraft II matches have been used to build agents that play using similar macro-management tactics as humans [65]. Having agents with realistic and human-like behaviour is important in simulation-based training as well. However, in the air combat domain there is very limited data available to support supervised learning. Another recent approach used soft Q-learning to constrain an agent’s policy to a reference policy, to create an agent whose performance could be adjusted to fit a human opponent [39]. In contrast to the method presented in this chapter, these methods do not consider multiple conflicting objectives. As identified in Chapter 5, having agents that can deal with multiple objectives is important in simulation-based air combat training.

In recent years there has been work on deep learning methods for multi-objective learning. Mossalam et al. used an outer-loop approach, which combined the single-objective DQN algorithm with a linear scalarisation function to calculate the set of optimal policies for convex combinations of the objectives [90]. A variation of optimistic linear support was used to select which utility function to optimise for in each iteration of the algorithm, and parameters from networks learned in past iterations were used as a starting point in later iterations, to improve efficiency. Roijers et al. presented a pol-
icy gradient algorithm for optimising the expected utility of the return in settings where the utility function of the user is known \[113\]. In later work, Hayes et al. extended learning with the ESR optimisation criterion to multi-policy learning \[48, 50, 49\]. Concurrently to our work, Abels et al. studied how methods based on deep Q networks could be used to successively learn new policies as users’ preferences among objectives are observed, either by constructing a set of networks or by using a single DQN conditioned on the user’s utility function \[3\]. In later works, utility and value conditioned networks have proven useful for efficient multi-policy learning \[96, 151, 163, 97, 107\]. Unlike the work by Abels et al., which studies the online setting of the dynamic utility function scenario (see Chapter \[3\]), we study an offline setting with the intention of providing decision support to the user. In addition, our work studies applications of multi-objective deep reinforcement learning in agent-based simulation, with a focus on interactive simulations for training, an area where not much work has been done. In this context, the work presented here studies how a single neural network representing an agent’s policy can be pre-trained for a broad range of preference weights, to represent agents with diverse characteristics. The approach is then evaluated in simulations in environments with discrete and continuous observation and action spaces, with single-agent as well as multi-agent learning and decision-making.

### 6.3 Method

This section presents an agent architecture and training scheme that allow the use of standard reinforcement learning algorithms in multi-objective environments. This results in a tunable actor called T-Actor, whose behaviour can be adjusted between different runs of an agent-based simulation to match user needs. The architecture, which is illustrated to the left in Figure 6.1, is a subset of the system architecture proposed in Chapter \[3\]. The agent presented in this chapter is a first realisation of our proposed Synthetic Trainer Agent.
To enable modeling that different agents have different goals and preferences over objectives, we include a reward system and a utility function as part of the agent. The reward system takes as input the agent’s observation of the environment, and outputs a vector reward signal $r_t = R(o_t)$, where each element in the vector represents the performance in one of the agent’s objectives. The utility function can be adjusted by an instructor, to set the agent’s relative preferences among its different objectives, in the form of objective weights $w$. In each time step the agent first observes the environment state, then it calculates a vector reward signal, and finally, based on its current preferences it calculates a corresponding scalar reward signal using its utility function $u$:

$$r = u(r, w) = \sum_{i=1}^{N} r_i w_i,$$

for a problem with $N$ objectives. This scalar is then used as the feedback signal when training the agent. One consequence of this approach is that different configurations of agents may have different opinions about which events and states are desirable and which are not, i.e. diversity among agents is supported. Since the utility function is applied to the rewards directly in each time step, we are restricted to using a linear function \[114, 115\].

The behavior of the agent is controlled by a policy produced by a learning algorithm. After applying the utility function to the vector reward the MOMDP has been scalarised to an MDP, and the input to the decision system and the learning algorithm is on a format that normal single-objective, value-based as well as policy-based algorithms can handle. In this chapter, we study settings where the policy of the agent is either represented by a Deep Q Network (DQN \[89\]) or a multi-agent actor-critic (MADDPG \[84\]), but other single-objective reinforcement learning algorithms and policy representations could be used. The observed state used as input to the learning algorithm is a combination of the environment state and the utility weights $w$ corresponding to the agent’s preferences among the objectives:

$$s_{tot} = [s_1, s_2, \ldots, s_M, w_1, w_2, \ldots, w_N].$$

This means that the agent will be trained to select actions according to the current selection of preferences among objectives. The observed state of the environment may be represented by a vector or a multi-dimensional tensor (e.g., an image). In the first case the state observation and the preference vector are simply concatenated and fed through a multi-layer perceptron (MLP) network. In the second case the multi-dimensional state representation is processed by a multi-layer convolutional neural network (CNN) before being flattened and concatenated with the preference vector, and then fed
through an MLP. The utility conditioned neural network of T-Actor is illustrated to the right in Figure 6.4. Depending on the learning algorithm used, the network can output an action distribution \( \pi \), a state value function \( V^\pi \), or a state-action value function \( Q^\pi \). Using a single neural network to represent these functions allows the agent to generalise across multiple preference weights, and efficiency can be improved compared to training a separate network for each utility function.

During training the agent must be exposed to a sufficiently large part of the set of possible preferences. Ideally, information from domain experts can help identify the relevant parts of the utility space, so that the full utility space does not have to be included in the search. The utility weight space is defined by two vectors, \( \text{pref}_{\text{high}} \) and \( \text{pref}_{\text{low}} \), specifying the maximum and minimum weight for each objective. This space is modelled as part of the agent, in a similar way as its action and observation spaces.

The training scheme for an off-policy tunable actor with experience replay is described in Algorithm 6.1. At the beginning of each episode the preference weight vector is initialised by sampling from a uniform random distribution according to the specified limits of the weight space for the agent, to promote exploration of the weight space. For each step in the episode, an action is chosen according to the current policy \( \pi(s_t, w) \), a reward vector \( r_t \) is observed, and the agent enters a new state \( s_{t+1} \). Examples from interactions with the environment are stored in an experience memory, after calculating a scalar reward using the utility function \( u \), corresponding to the current preference weights. Periodically, a batch of experiences is sampled from the experience memory, and used to update the policy of the agent.

6.4 Experimental Evaluation

This section presents the experimental evaluation of T-Actor. We begin with evaluations in gridworlds, and then continue to validating experiments in the target system. We study the number of time steps required for the agent to converge, as well as the agent’s qualitative behaviour when configured with different utility functions. Experimental results are averaged over five iterations with different seeds for random number generators.

Gridworld Environments

As an initial evaluation of the proposed method experiments are conducted in two multi-agent gridworlds. The gridworlds are designed to study two categories of environment characteristics that are of interest in air combat training: tunable competitiveness and tunable risk-taking. In these environments, we do not only parameterise the utility function of the agent, but also its reward signal. We do this by providing the learning agent with an observation of events that occur in the environment in each time step. Each event
6.4. Experimental Evaluation

Algorithm 1 Training scheme for off-policy tunable actor

1: procedure TrainAgent
2: \( t_{total} \leftarrow 0 \)
3: while \( t_{total} < t_{max} \) do
4: \( obs \leftarrow env.reset() \)
5: \( w \leftarrow preference\_space.sample() \)
6: \( done \leftarrow False \)
7: for \( t = 1 \) to episode\_length do
8: \( act \leftarrow agent.act(obs, w) \)
9: \( obs\_new, done \leftarrow env.step(act) \)
10: \( r \leftarrow R(obs\_new) \)
11: \( memory.store(obs, w, act, u(r, w), obs\_new) \)
12: \( obs \leftarrow obs\_new \)
13: \( t_{total} \leftarrow t_{total} + 1 \)
14: if \( (t_{total} \mod \text{train\_frequency}) = 0 \) then
15: \( batch \leftarrow memory.sample() \)
16: \( policy.update(batch) \)
17: if \( done \) then
18: break

is related to one of the objectives of the agent. The agent’s preference weight vector is then used to define what value is associated with each event. Unlike the standard approach of scalarisation using a convex combination of the elements of a vector reward provided by the environment, we allow preference weights to have negative as well as positive values, and the weights do not need to sum to one. This results in increased flexibility when modelling agent behaviour.

The size of each environment grid is 8x8 cells. The agent’s action space allows it to take the actions Up, Down, Left, Right, or Stay. The transition dynamics of the agent are deterministic, while parts of the environment contents are randomised for each episode. Agents receive visual observations of the environment consisting of the last four frames as stacked RGB arrays. For the gridworld scenarios, the agents are configured to use a DQN with three initial convolutional layers for learning state features. The following network design is used, where each tuple states number of filters, kernel size, and stride for a layer: \([16, 3, 1], (32, 3, 1), (64, 2, 1)\]. The flattened output from the feature learning layers is concatenated with the agent’s utility function and fed into a duelling head for learning \(Q\) values. Each branch of the duelling head is configured with an MLP, which has one hidden layer with 256 neurons. We use the ReLU activation function in all layers. The agents are trained with batches of 32 samples from the experience replay buffer, which has room for 100,000 samples and uses prioritised experience replay. In training, we use a learning rate of \(\alpha = 10^{-4}\), discount factor \(\gamma = 0.99\), and the Adam op-
A vector of weights is used to represent the agent’s preferences over objectives. We use the implementation of the DQN learning algorithm available in the OpenAI Baselines repository.

After training the tunable actors we used them as a decision-support mechanism, by running simulations and adjusting the preference weights within the specified weight space in search of distinct behaviours. For the found interesting weight settings we trained agents with fixed objective preferences for the same number of episode steps as the tunable actors. The relative performance of the two types of agents was then evaluated in simulations with 10,000 episodes for each preference setting.

**Item Gathering Environment**

The first gridworld environment is an Item Gathering environment, which is intended to study how the proposed method can be used to tune agents to act more or less competitively with respect to other agents. Such functionality could be valuable in training and gaming scenarios, to create agents that are suitable for the level of a human opponent. In air combat training, changes in competitiveness could be used to adjust the challenge of training scenarios as trainees move from basic training to mission training. The studied environment contains different types of items that the agent should collect. There are three green items, three red items, and two yellow items. Before the start of each episode, the items are placed by random in the 16 cells in the centre of the grid. In addition to the learning agent, there is also another agent acting in the environment, with fixed preference for collecting red items. The tunable actor is trained to be able to prioritise among red, green and yellow items.
items at runtime. The agent also has a tunable preference for other agents collecting items. In this way it is possible to tune to which extent this agent should be competitive or cooperative in relation to other agents. This means that the agent could act in several of the roles we discussed in the user study of Chapter 3. In addition to the tunable preferences, there is also a fixed penalty for each step in the environment and for trying to walk outside the grid. An episode ends after 30 steps or when all items preferred by some agent have been collected. An example of the environment is shown in Figure 6.3. For this environment we train with 1 million episode steps. For exploration $\epsilon$-greedy action selection is used, i.e. with probability of $\epsilon$ a random action is selected. $\epsilon$ is decayed from 1 to 0.01 over 15% of the training steps.

For this environment the list of tunable objectives is [green item collected, red item collected, yellow item collected, items collected by other agent], and we use a preference weight space defined as:

$\text{pref}_{\text{high}} = [+20, +20, +20, +20]$  
$\text{pref}_{\text{low}} = [-20, -20, -20, -20]$  

When we studied the tunable actor’s behaviour for different preference weights in the Item Gathering environment, we found four scenarios for the following sets of preferences over the objectives:

- Scenario 1, Competitive Agent, $w = [+10, +20, +10, -20]$: Here the agent has a negative preference for other agents collecting items, while red items are valued the most. With these preferences it is expected that the tunable actor will exhibit competitive characteristics, and that the hard-coded agent will struggle to collect items.

- Scenario 2, Cooperative Agent, $w = [+10, +20, +10, +20]$: In contrast to the previous scenario, in this scenario the agent also has a positive preference for other agents collecting items. With these preferences it is expected that the hard-coded agent will have a better chance of collecting items.

- Scenario 3, Fair Agent, $w = [+20, +15, +20, +20]$: Here the agent has a preference for green and yellow items over red ones, and also has a positive preference for other agents collecting items. With these preferences it is expected that the tunable actor will leave items for the other agent to collect.

- Scenario 4, Generous Agent, $w = [+20, +0, +20, +20]$: The agent receives high rewards for collecting green and yellow items, no rewards for red items, and high rewards if other agents collect red items. With these preferences it is expected that the tunable actor will avoid collecting red items, and instead leave them for the other agent.
The training progress of the tunable actor and agents with fixed objective preferences for the identified scenarios in the Item Gathering environment is shown in Figure 6.3, as mean 100 episode return over the episodes of training. The tunable actor converges after approximately 40,000 episodes, while the agents learning with fixed utility functions converge after approximately 20,000 episodes, although some slight improvements in performance can be seen towards the later stages of training. For the agents with fixed utility functions, the learning curves for scenarios 3 and 4 almost overlap.

The results of the simulations with the trained agents in the identified scenarios are presented to the right in Figure 6.3, as average utility achieved, and in Figure 6.4 as the average number of items collected for each category. In Figure 6.3 we can see that the quantitative performance of agents trained with fixed preferences is slightly better than that of the tunable actor. The biggest difference is for the competitive setting, which is the most challenging one. In Figure 6.4 we can see that the qualitative behaviour of the tunable actor is a good approximation of the behaviour of agents with fixed utility functions. Moving from scenario 1 to scenario 4, i.e. from competitive to generous behaviour, the tunable actor focuses less on collecting red items, allowing the other agent to collect more of those items. This change in focus also results in slightly better performance in collecting green and yellow items.

Overall, it can be seen that the tunable actor and the set of fixed preference agents produce similar quantitative results and have the same qualitative behaviour, even though the tunable actor has been trained with fewer time steps than the total number of time steps used for training of the set of fixed preference agents. In addition, the tunable actor does not require users to provide an exact reward signal or utility function a priori. In real world settings, the initial policy set found by the tunable actor could be presented to
### 6.4. Experimental Evaluation

#### Scenario 1

<table>
<thead>
<tr>
<th></th>
<th>T-Actor</th>
<th>Fixed Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>2.89</td>
<td>2.98</td>
</tr>
<tr>
<td>red</td>
<td>1.35</td>
<td>1.47</td>
</tr>
<tr>
<td>yellow</td>
<td>1.92</td>
<td>1.98</td>
</tr>
<tr>
<td>other's</td>
<td>1.65</td>
<td>1.53</td>
</tr>
</tbody>
</table>

#### Scenario 2

<table>
<thead>
<tr>
<th></th>
<th>T-Actor</th>
<th>Fixed Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>2.98</td>
<td>3.0</td>
</tr>
<tr>
<td>red</td>
<td>0.76</td>
<td>0.76</td>
</tr>
<tr>
<td>yellow</td>
<td>1.99</td>
<td>2.0</td>
</tr>
<tr>
<td>other's</td>
<td>2.24</td>
<td>2.24</td>
</tr>
</tbody>
</table>

#### Scenario 3

<table>
<thead>
<tr>
<th></th>
<th>T-Actor</th>
<th>Fixed Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>2.98</td>
<td>2.99</td>
</tr>
<tr>
<td>red</td>
<td>0.39</td>
<td>0.08</td>
</tr>
<tr>
<td>yellow</td>
<td>1.98</td>
<td>2.0</td>
</tr>
<tr>
<td>other's</td>
<td>2.61</td>
<td>2.92</td>
</tr>
</tbody>
</table>

#### Scenario 4

<table>
<thead>
<tr>
<th></th>
<th>T-Actor</th>
<th>Fixed Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>2.99</td>
<td>2.99</td>
</tr>
<tr>
<td>red</td>
<td>0.17</td>
<td>0.03</td>
</tr>
<tr>
<td>yellow</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>other's</td>
<td>2.82</td>
<td>2.97</td>
</tr>
</tbody>
</table>

Figure 6.4: Average number of different categories of items collected in different configurations of the Item Gathering environment. From top row to bottom row: green items, red items, yellow items, and other agent’s items. From left column to right column: Scenario 1 to scenario 4. Brighter colours indicate higher values. **Left:** Metrics for tunable actor. **Right:** Metrics for agents with fixed objective preferences.

subject matter experts, to define a new, smaller utility space for additional learning and improved performance in the most useful scenarios.

**Traffic Navigation Environment**

The second gridworld environment is a Traffic Navigation environment, which is intended to study how the proposed method can be used to tune the risk-awareness of agents. Such functionality could be valuable in traffic simulations as well as military simulations. In air combat simulation, agents that are willing to take risks can use a more offensive tactic, and put pressure on the human trainees. The environment contains two green items that the agent needs to collect. These items are placed in the two upper corners of the grid. In the centre of the grid there is a yellow road segment where scripted red agents, representing cars, are moving vertically. When a car hits a wall (white cell in the grid) or the edge of the grid it changes direction. At the beginning of each episode the initial position and direction, as well as the speed of each car are selected by random. According to the traffic rules the tunable actor is not allowed to pass through the road segment, but can choose to do so in violation of traffic rules if it pleases, and if it is willing to risk colliding with a car. The agent must balance rule abidance and the risk of passing the road segment against the time it takes to collect the two items. In addition to the tunable preferences, there is also a fixed penalty for trying to walk outside the grid or into a wall. An episode ends after 50 steps or when both green
items have been collected by the agent. An example of the environment is shown in Figure 6.5. This environment is more challenging than the Item Gathering environment, so the training time is extended to 10 million episode steps. More exploration is also used, with $\epsilon$ decaying from 1 to 0.01 over 30% of the training steps.

For this environment the list of tunable objectives is [steps, item collected, steps on road, collisions], and we use a preference weight space defined as:

$$ pref_{\text{high}} = [-1, +50, 0, 0] $$

$$ pref_{\text{low}} = [-10, +50, -20, -50] $$

When we studied the tunable actor’s behaviour for different preference weights in the Traffic Navigation environment, we found four scenarios for the following sets of preferences over the objectives:

- **Scenario 1, Always Safe**, $w = [-1, +50, -20, -50]$: Here there is a big penalty for passing the road and colliding with cars. This configuration should encourage the agent to take the long route around the road segment, and thus avoid collisions with cars.

- **Scenario 2, Always Fast**, $w = [-10, +50, -10, -10]$: Here there is a big time penalty, but also penalties for walking on the road segment or colliding with cars. We set the penalty for walking on the road segment to the same as for colliding with cars, so that the agent may deliberately walk into a car to avoid having to take a costly detour.

- **Scenario 3, Fast and Safe**, $w = [-5, +50, 0, -50]$: Here there is a medium time penalty and a high penalty for colliding with cars. However, there
is no penalty for walking on the road segment, so the agent is encouraged
to take a shortcut to reach the end of the episode quickly.

- Scenario 4, Patient and Safe, $w = [-1,+50,0,-50]$: Here there is a small
time penalty and a high penalty for colliding with cars, but no penalty
for walking on the road segment. With this prioritisation the agent is
expected to try to take a shortcut through the road segment if the traffic
conditions are favourable, but otherwise be patient and find a longer,
safer route.

The training progress of the tunable actor and agents with fixed objective
preferences for the identified scenarios in the Traffic Navigation environment
is shown in Figure 6.6, as mean 100 episode return over the episodes of training.
The tunable actor converges after approximately 100,000 episodes, as does the
agent with fixed preferences for being fast as well as safe (scenario 3). Agents
with fixed preferences according to scenarios 1 and 4 converge slightly faster,
after roughly 80,000 episodes, and their returns then almost overlap. The
agent with fixed preferences for always being fast takes the longest to converge,
after roughly 110,000 episodes, and has high variance over the iterations of
learning.

The results of the simulations with the trained agents in the identified
scenarios are presented to the right in Figure 6.6, as average utility achieved,
and in Figure 6.7 as the average number of events that occurred for each
category. In addition, Figure 6.8 shows the routes most frequently selected
by the agents in each scenario. In Figure 6.6 we can see that the quantitative
performance of the tunable actor is close to that of the set of agents trained
with fixed preferences in all scenarios. The biggest difference is for scenario 4,
where agents have time to manoeuvre safely around vehicles in the traffic grid.
Figure 6.7 and Figure 6.8 show that the qualitative behaviour of the tunable
actor is a good approximation of the behaviour of agents with fixed utility.
6. Tunable Dynamics in Agent-Based Simulation

<table>
<thead>
<tr>
<th></th>
<th>T-Actor</th>
<th>Fixed Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>steps</td>
<td>28.88 14.05 20.11 23.27</td>
<td>28.0 14.13 19.35 20.91</td>
</tr>
<tr>
<td>objects</td>
<td>2.0 2.0 2.0 2.0</td>
<td>2.0 2.0 2.0 2.0</td>
</tr>
<tr>
<td>violations</td>
<td>0.0 6.0 6.57 6.63</td>
<td>0.0 6.0 7.24 7.29</td>
</tr>
<tr>
<td>collisions</td>
<td>0.0 1.8 0.3 0.14</td>
<td>0.0 1.73 0.32 0.06</td>
</tr>
</tbody>
</table>

Figure 6.7: Average number of different categories of events occurring in different configurations of the Traffic Navigation environment. From top row to bottom row: steps taken, objects collected, traffic violations committed, and collisions with cars. From left column to right column: Scenario 1 to scenario 4. Brighter colours indicate higher values. **Left:** Metrics for tunable actor. **Right:** Metrics for agents with fixed objective preferences.

functions. In scenario 1, the agents always take a safe route around the traffic grid, and no traffic violations or collisions are registered. In scenario 2, the agents have a strong focus on completing the episode as quickly as possible, by moving along the top of the traffic grid, which results in an increase in collisions. In scenarios 3 and 4, agents try to find a better balance between time and safety, by passing through the centre of the traffic grid grid while manoeuvring to avoid collisions. This results in several traffic violations, but few collisions.

Overall, we noted that scenario 2 caused the most problems for agents learning with fixed preferences over objectives, as illustrated by the longer training time to conversion and worst relative final performance compared to the tunable actor. The tunable actor did not struggle in the same way for this setting, even though it learned competitive policies for the other scenarios in parallel. This illustrates the value of using a single neural network for representing the agent’s behaviour for different utility functions. The experiences gathered when learning a policy for one utility function are valuable when learning policies for other utility functions as well.

**Risk-Aware Flight Route Selection**

This section studies the performance of the tunable actor in a more challenging scenario of Risk-Aware Flight Route Selection. Having an ability to find suitable routes is important in all phases of air combat. In the studied scenario, a synthetic pilot must reach a target location in an attack mission,
Figure 6.8: Routes in the *Traffic Navigation* environment for tunable actor and agents with fixed objective preferences. Brighter areas have been visited more often.

while avoiding enemy air defence systems. The agent must prioritise between time and safety for maximum utility when selecting a route. In each episode the agent starts in the same position, with initial heading towards the target, and one air defence system placed between the agent’s start position and the target. The agent can observe the air defence system’s position through the aircraft’s sensors. The agent must learn to manoeuvre the aircraft along a suitable route in relation to the threat for the current utility function. Unlike the *Traffic Navigation* environment, the agent is now acting in a continuous state space, which adds additional challenge when trying to find optimal routes. The scenario is illustrated in Figure 5.4. The star represents the aircraft starting position, the red circle illustrates the range of the air defence system, and the green circle represents the target area. The displayed geographical region has a size of 100 km × 100 km. As for the scenarios studied in Chapter 4, we implemented this scenario in the GYM-CGF reinforcement learning environment, and ran the simulations in the tactical environment simulation of an operational pilot training system.

The DQN implementation from OpenAI Baselines [28] is used as the core learning algorithm, as in the gridworld scenarios. The policy is represented by an MLP, with 2 initial hidden layers with 64 neurons each for feature extraction. These layers are followed by a duelling head with 1 hidden layer with 256 neurons in each branch for learning Q values. We use the ReLU activation function in all layers. We use a learning rate of $\alpha = 10^{-4}$, a discount factor of $\gamma = 1.00$, a replay-buffer with 200,000 samples, and train using prioritised experience replay and the Adam optimiser [76]. $\epsilon$-greedy action selection is used.
for exploration, with $\epsilon$ decaying from 1 to 0.01 over 10% of the training steps. We then compare the performance of the tunable actor with the performance of agents learning with fixed preferences between time and safety.

The observation space of the agent is the relative bearing and distance to the centre of the threat area, and the relative bearing and distance to the target, for the last 4 time steps in the episode. The elements of the observation vector are normalised by their expected maximum value. Since DQN does not handle continuous actions, we must discretise the commands to the lower level controllers of the simulated aircraft. Therefore, we define the agent’s action space as forward motion or right or left turns with a load factor of 2-4 g in discrete steps of 1 g. The agent selects actions at 4 s intervals, and the selected action is repeated every 1 s step in between. The episode ends if the agent reaches the target, or terminates after a maximum of 400 time steps. The reward vector is defined as:

$$r_t = [r_{time}(t), r_{ad}(t), r_{goal}(t)],$$

(6.3)

$$r_{time}(t) = -0.5,$$

(6.4)

$$r_{ad}(t) = \begin{cases} -\left( \frac{1}{10}(R_{ad} - d_{ad}(t)) \right)^2 & \text{if } d_{ad}(t) \leq R_{ad}; \\ 0 & \text{if } d_{ad}(t) > R_{ad}; \end{cases}$$

(6.5)

$$d_{ad}(t) = \|p_{ad}(t) - p_a(t)\|,$$

(6.6)
6.4. Experimental Evaluation

\[ r_{\text{goal}}(t) = \gamma \phi(s_t) - \phi(s_{t-1}) = d_g(t-1) - \gamma d_g(t), \]  
\[ (6.7) \]

\[ d_g(t) = \|p_g(t) - p_a(t)\|, \]  
\[ (6.8) \]

where \( r_{\text{time}}(t) \) refers to the reward for the objective of reaching the target fast, \( r_{\text{ad}}(t) \) refers to the reward for the objective of avoiding being attacked by the air defence system, and \( r_{\text{goal}}(t) \) refers to a potential-based reward shaping \([94, 75, 87]\) element in the reward vector. The reward shaping rewards the agent for reducing the distance to the target, and thus guides the exploration of the agents. If we were to set \( \phi(s_0) = \phi(s_T) = 0 \) for the initial state \( s_0 \) and terminal state \( s_T \), the shaping rewards would cancel out in the episodic return \([94, 10, 87]\), and have no influence on the learned policy. With our design of the shaping reward, there will be a contribution left in the return. However, in practice we could see that using the same form of shaping in all states still resulted in a significant improvement of performance compared to learning without a shaping reward. \( p_a(t) \), \( p_g(t) \) and \( p_{\text{ad}}(t) \) are the positions of the aircraft, goal and air defence system, \( d_g(t) \) and \( d_{\text{ad}}(t) \) are the distances from the aircraft to the goal and air defence system, and \( R_{\text{ad}} = 20 \) km is the range of the air defence system. The risk-related reward in Equation 6.5 models that risk increases more rapidly as the aircraft moves deeper into the threat area. The scaling of the reward is intended to result in comparable magnitude of the rewards given for the time and risk objectives.

To specify the priorities of the Time and Safety objectives, we use the weight \( \theta \). We then use the following scalarisation function to convert the vector reward to a scalar (with \( \theta \in [0.65, 1.00] \) specifying the utility space of the agent):

\[ r_t = \theta \cdot r_{\text{time}}(t) + (1 - \theta) \cdot r_{\text{ad}}(t) + r_{\text{goal}}(t) \]  
\[ (6.9) \]

For the tunable actor we sample \( \theta \) from a uniform distribution over \([0.65, 1.00]\) before each episode and use it as input to the agent, while training for 2 million time steps (8 million time steps including action repeat). For comparison, we train fixed policy agents for the same number of time steps, for \( \theta \in \{0.75, 0.85, 0.95\} \). The training progress of tunable actors and fixed policy agents is presented in Figure 6.10, averaged over five runs with different random seeds.

We can see that training is somewhat unstable, with spikes of high variance. The cause may be that small changes in policy have great effect on the accumulated reward, or that is difficult for the agent to learn the characteristics of the reward function. It is also possible that the low frequency of the controller or the discretisation of the action space has a negative effect on
6. **Tunable Dynamics in Agent-Based Simulation**

Figure 6.10: **Left:** Training progress for tunable actor. **Right:** Training progress for a set of fixed agents with different priorities among objectives.

Figure 6.11: Average accumulated rewards for individual objectives in the *Risk-Aware Flight Route Selection* environment for tunable actor and agents with fixed objective preferences. Brighter areas correspond to higher rewards. **Left:** Performance of the tunable actor. **Right:** Performance of agents with fixed objective preferences.

performance. The learning progress of the tunable actor starts to plateau after roughly 5000 episodes, with some disturbances until 10,000 episodes, and followed by some slight improvements in later episodes. Agents learning with fixed preferences over objectives corresponding to \( \theta = 0.75 \) and \( \theta = 0.95 \) also start to plateau at around 5000 episodes, while the agents learning with fixed preferences over objectives corresponding to \( \theta = 0.85 \) keep improving slightly to 10,000 episodes, where there are some disturbances in the learning curves.

Results for each objective in simulations after training, for preferences corresponding to high, medium and low risk exposure, are presented in Figure 6.11. It can be seen that when \( \theta \) decreases, more priority is given to the safety objective. It can also be seen that the biggest difference in behaviour between the tunable actor and agents with fixed objective preferences is for
θ = 0.95, i.e. the utility function that prioritises time over safety. The mean and standard deviation for the utility (without the shaping reward) accumulated by tunable and fixed policies in the simulations are presented to the left in Figure 6.12. We can see that the tunable actor produces competitive results for θ = 0.85 and θ = 0.75, but performs worse for θ = 0.95. The worse result for the tunable actor is highly affected by one of the five training iterations, which performs significantly worse than the remaining four. The agent in this iteration selects a safe route even though the priority is on time. This leads to high standard deviation for this case.

To further study what policies the tunable actors had learned during training, we ran simulations where θ was increased from θ = 0.65 to θ = 1.00, in steps of 0.01. We used the vector returns achieved by all five tunable actors to form approximate Pareto and convex coverage sets (with 36 and 10 points respectively), as illustrated to the right of Figure 6.12. The CCS, which contains the policies that are optimal for the linear utility function that we use, is fairly wide-spread over the value space for the time and safety objectives. We can see that there are policies to the left of the plot that almost avoid risk completely, by flying a longer route around the threat area. To the right of the plot there are value vectors corresponding to policies that accept high risk to save time. The diverse set of policies captured in the utility conditioned networks would allow an instructor to adapt training scenarios for increased variation and improved training effect. As identified in the user study presented in Chapter 3, in air combat training we do not necessarily want agents to perform optimal actions in every step of the simulation. Instead they should sometimes make small mistakes, which the trainees can try to exploit. One way to achieve this could be to periodically switch to a policy that lies outside of the CCS, but still results in credible behaviour. The density of the front is better in the middle parts than at the edges. This is probably a result of the
uniform sampling of the utility space, which results in the agent encountering more utility functions that are relevant for balanced policies that have similar priorities for time and risk management.

Three simulated flight routes selected for \( \theta \in \{0.75, 0.85, 0.95\} \) are illustrated in Figure 6.13, for the tunable actor and fixed policies. For the agents learning with fixed preferences over objectives, the routes presented correspond to those out of the five iterations that achieve the highest utility, while the routes for the tunable actor all come from one learning agent, which has good overall performance. For lower values of \( \theta \) agents select routes further from the threat, while for higher values of \( \theta \) agents select routes with higher risk. We can see that the manoeuvring of the tunable actor is not as exact as that of agents with fixed preferences over objectives, especially so for the route corresponding to \( \theta = 0.95 \). In this scenario, routes can be adjusted continuously from the centre of the threat area to its outer edges, which makes it challenging to find the optimal routes for all utility functions. More exploration could help improve the performance of the tunable actor.

**Risk-Aware Air Interdiction with Fighter Escort**

Finally, the performance of the tunable actor has been evaluated in an *Air Interdiction with Fighter Escort* scenario, which requires multi-agent learning. In air combat, missions are always carried out in groups, and having the ability to cooperate to reduce risks is important. In this scenario, a strike aircraft should navigate to a target location protected by an air defence sys-
6.4. Experimental Evaluation

Figure 6.14: Simulation scenario for Air Interdiction with Fighter Escort.

A team is formed with one strike aircraft and one escort aircraft, which should protect the strike aircraft from the ground-based threat while the team approaches the target area. The escort aircraft is equipped with a jamming pod, which can be used to reduce the performance of a radar system co-located with the air defence system, so that the two approaching aircraft can not be positioned. This requires that the aircraft fly in a formation, aligned with the same heading.

The scenario is illustrated in Figure 6.14. The displayed geographical region has a size of 100 km × 100 km. In each episode, the target location is initialised in a random position within a blue rectangular area to the north, while the two aircraft are initialised in random positions with random heading, within a green rectangular area to the south. Example initial positions of target (black circle) and aircraft (black stars) in one episode are shown in the figure. The range of the air defence system co-located with the target is shown by the red circle.

As in the Risk-Aware Flight Route Selection environment, we implement this environment using the GYM-CGF framework and the tactical environment simulation of a pilot training system. In the observation space of the agents, positions are given in a body-fixed coordinate system, while headings are given relative true north, just as in the scenarios studied in Chapter 4, and as illustrated in Figure 4.1. Episodes end if the strike aircraft comes within 10 km of the target, or after a maximum of 300 time steps. Each time step is 1 second long. MADDPG is used as the core learning algorithm to train the agents. We use a learning rate of $\alpha = 10^{-2}$, a discount factor of $\gamma = 0.90$, a replay-buffer with 1 million samples, and train for 50,000 episodes using the Adam optimiser [76]. The policy is represented by an MLP with 2 hidden layers, each with 64 neurons and the ReLU activation function.
6. Tunable Dynamics in Agent-Based Simulation

As in the Risk-Aware Flight Route Selection environment, we want the agents to consider two conflicting objectives: time and safety. This goal is modelled with a shared vector reward signal defined as:

\[ r_t = [r_{\text{time}}(t), r_{\text{safe}}(t), r_{\text{tgt}}(t)] \] (6.10)

Here \( r_{\text{time}} \) and \( r_{\text{safe}} \) are the rewards given for the time and safety objectives respectively. The time objective is achieved by reaching the target as quickly as possible. The safety objective is achieved by flying in a way that enables efficient use of the escort aircraft’s jamming pod, i.e. flying in close formation and aligning towards the target. The rewards given for each of the objectives are defined as:

\[ r_{\text{time}}(t) = -1.0 \] (6.11)

\[ r_{\text{safe}}(t) = r_{\text{form}}(t) + r_{\text{align}}(t) \] (6.12)

As in the Risk-Aware Flight Route Selection Scenario, in addition to the two major objectives, we also use a potential-based reward shaping [94, 87] element in the reward vector \( r_{\text{tgt}} \), which rewards the agents for reducing the distance between the strike aircraft and the target \( d_{\text{tgt}} \):

\[ r_{\text{tgt}}(t) = d_{\text{tgt}}(t-1) - \gamma d_{\text{tgt}}(t) \] (6.13)

To encourage the agents to fly in close formation, they are given a penalty proportional to the distance between the escort aircraft and a formation reference point relative the strike aircraft:

\[ d_{\text{form}}(t) = ||p_{\text{escort}}(t) - p_{\text{ref}}(t)|| \] (6.14)

\[ r_{\text{form}}(t) = -0.2 \cdot d_{\text{form}}(t) \] (6.15)

Here \( p_{\text{escort}} \) and \( p_{\text{ref}} \) are the positions of the escort aircraft and the formation reference point respectively, \( d_{\text{form}} \) is the distance between the escort and its reference point, and \( r_{\text{form}} \) is the reward component given for flying in close formation.

To encourage the agents to align towards the target, they are given a reward dependent on the alignment error, when the positions of the strike aircraft and the target \( (p_{\text{strike}} \text{ and } p_{\text{target}}) \) are both within the Field-of-View (FOV) of the escort aircraft’s jammer:

\[ jam_\delta(t) = jam_\delta(t) + jam_{\delta_{\text{tgt}}}(t) \] (6.16)

\[ r_{\text{jam}}(t) = 0.5 \cdot (jam_{\text{fov}}(t) - jam_\delta(t))/jam_{\text{fov}}(t) \] (6.17)
6.4. Experimental Evaluation

\( r_{\text{align}}(t) = \begin{cases} r_{\text{jam}}(t) & \text{if } p_{\text{strike}} \text{ and } p_{\text{tgt}} \text{ in } \text{jam}_\text{fov} \\ 0 & \text{ELSE} \end{cases} \) (6.18)

Here \( \text{jam}_\text{fov} \) is the FOV of the escort aircraft’s jammer, \( \text{jam}_\delta \) and \( \text{jam}_\delta_{\text{tgt}} \) are the absolute angular alignment errors between the jammer centre line and the positions of the strike aircraft and target respectively, and \( r_{\text{jam}}(t) \) is a reward component given for using the jammer effectively. \( r_{\text{align}}(t) \) is the reward component given for aligning towards the target.

In these experiments, the formation reference point for the escort aircraft was placed 5 km behind the strike aircraft, on a line passing through the position of the strike aircraft and the position of the target. The jammer FOV was set to 60 degrees. The scale factors in the reward components were selected to give the components comparable magnitude for this scenario, i.e. no component should completely dominate the others.

To specify the priorities of the time and safety objectives, we use the weight \( \theta \). We then use the following scalarisation function to convert the vector reward to a scalar (with \( \theta \in [0.2, 0.8] \) specifying the utility space of the agent):

\[ r_t = \theta \cdot r_{\text{time}} + (1 - \theta) \cdot r_{\text{safe}} + r_{\text{tgt}} \] (6.19)

The resulting scalar reward \( r_t \) is then used as input to MADDPG for training of the agent’s policy. As in the Risk-Aware Flight Route Selection scenario, \( \theta \) is included in the input to the agent’s policy, so that it can be used to adjust the agent’s behaviour after training, i.e. we want the agent to learn how the value of \( \theta \) affects its reward. To allow the agents to learn how to prioritise between time and safety, we sample \( \theta \) from a uniform distribution over the interval \([0.2, 0.8]\) before each episode of training.

In each time step an agent observes the observation vector \( o_t \), which contains its own heading and speed, the position, heading and speed of the other agent, the position of the formation reference point, the position of the target, and the preference between time and safety objectives (\( \theta \)). All elements of the observation vector \( o_t \) are normalised by their expected maximum value. The complete observation space of each agent, which is the input to the neural network representing the agent’s policy, is the set of observations from the last four time steps:

\[
O_t = \begin{bmatrix} o_t \\ o_{t-1} \\ o_{t-2} \\ o_{t-3} \end{bmatrix},
\] (6.20)

We use a tuple action space:
\[ A = A_{goal} \times A_{thrust} \] (6.21)

\(A_{goal}\) is a discrete, two element action space, which allows the agents to move towards each other (the strike aircraft moving towards its escort, the escort moving towards its reference point), or to move towards the target. \(A_{thrust}\) is a continuous action space, which allows the agents to set the commanded speed in the interval Mach \([0.4,1.2]\).

After training, we evaluate the approach by studying the trained agents’ behaviour in 10,000 simulations for three different values of \(\theta\) corresponding to Safe, Balanced and Fast policies: \(\{0.2,0.5,0.8\}\). For comparison, we train and evaluate the same type of agents using the single-agent reinforcement learning algorithm DDPG, as well as MADDPG agents with fixed utility functions for each of the three values of \(\theta\).

The training progress of tunable actors trained with MADDPG and DDPG is shown to the left in Figure 6.15, as mean and standard deviation of the return achieved for the shared reward signal. We can see that the agents trained with MADDPG improve quickly and start to plateau already after 10,000 episodes, while the agents trained with DDPG need more than 40,000 episodes to reach similar performance. This indicates that the studied scenario actually requires multi-agent coordination, and that using a dedicated multi-agent learning algorithm in addition to the shared reward of the agents improves performance. The results of the DDPG agents also have higher variance, and the mean rewards dip towards the end of training, which may be explained by the non-stationarity resulting from multiple agents learning and updating their policies within the same system.

The training progress of agents with fixed objective preferences is shown to the right in Figure 6.15, as mean and standard deviation of the return achieved for the shared reward signal. We can see that there is slightly less variance than for the tunable actors, especially compared to those using DDPG for learning. The agents prioritising reaching the target Fast converge quickly, after roughly 5000 episodes. Agents prioritising Balanced or Safe behaviour take longer to converge, and stabilise towards 20,000 episodes. Towards the end of training, the performance of the Safe agent drops while the performance of the Balanced agent improves, and they end at roughly the same mean return. In the studied scenario, the safer policies require more cooperation between agents, which is reflected in the observed training performance of the agents with fixed utility functions.

The results of simulations with three different values of \(\theta\) for 10,000 episodes are shown in Figure 6.16. We can see that the qualitative results are as desired for the tunable MADDPG agents: When \(\theta\) is small, priority is given to the safety objective, when \(\theta\) increases, more priority is given to the time objective. This effect is present for the tunable DDPG agents as well, but it is not as prominent. For the MADDPG agents with fixed utility
functions, the performance in the safety objective is better for $\theta = 0.5$ than for $\theta = 0.2$, which is related to the changes in learning performance for these agents noted above.

The achieved utility for different utility functions (without the shaping reward) is presented to the left of Figure 6.17. We can see that tunable DDPG policies perform increasingly worse compared to MADDPG policies as the value of $\theta$ decreases and cooperation becomes more important, showing again the challenge of learning cooperation with independent learning. The significance of differences in performance between tunable MADDPG and MADDPG with fixed utility functions is small. For $\theta = 0.8$, the tunable actor outperforms the agent learning with a fixed preference for completing the episode in short time. It is possible that the tunable actor could find a better trade-off between time and safety thanks to its exploration of the environment with multiple utility functions. To the right of Figure 6.17, the emergent behaviour of tunable actors after training is shown in an example episode. Trajectories for safe agents ($\theta = 0.2$) are shown in green and trajectories for
fast agents ($\theta = 0.8$) are shown in red. The escort aircraft starts in the bottom left corner of the spawn area for this episode, while the strike aircraft starts in the upper right corner. Safe agents align before entering the range of the air defence system (taking 179 seconds to finish the episode), while fast agents enter the risk area before aligning properly (taking 141 seconds to finish the episode).

## 6.5 Summary

In this chapter we have proposed an agent architecture and training scheme intended for agent-based simulation. The architecture allows us to use standard deep reinforcement learning algorithms in multi-objective environments. The proposed approach can be used to train agents whose behaviour can be adjusted at runtime, by specifying the agents’ preferences among a set of objectives. Our experiments demonstrate that these tunable actors can approximate the policies of several different behavioural categories of agents with fixed preferences among objectives. The experiments also show that the training time is comparable to that of agents with fixed preferences, meaning that sample efficiency can be improved compared to learning a set of policies from scratch. Once interesting regions of the solution front have been found, for instance in interaction with subject matter experts, policies in those regions can be improved upon through further training. We argue that this functionality is highly valuable for efficient and effective construction of agent-based simulations adapted to user needs, e.g. for application in training systems.

One downside with the linear utility function we have used is that if a deterministic policy is used and the problem has a concave Pareto front all desired policies may not be found \cite{141, 144}. To get better coverage of the solu-
tion space, a non-linear scalarisation function must then be used. If stochastic policies are allowed, convex mixtures of policies from the CCS can be used to form a continuous estimate of the Pareto front, which dominates all solutions in concavities \[141\]. However, this may not be useful if we are interested in the outcome of each episode (ESR), rather than the average outcome of multiple episodes (SER). For user-adaptive simulation-based training the ESR setting would be the suitable one, as discussed in Chapter 5.

The environments we have studied are relatively simple. For real-world scenarios it could be more challenging to train this type of agent. In future work we would like to study the performance of the proposed method in more complex environments, including environments with large action spaces, partial observability, and tasks that require more complex interactions among agents. We would also like to study intelligent exploration strategies, that allow agents to be trained with many objectives and preference spaces with high dimension, as well as efficient transfer learning to unseen weights. Another interesting topic for future work is development of efficient methods for elicitation of user preferences regarding agent characteristics, to enable construction of simulations that match user needs. Finally, we would also like to study human-agent interaction in e.g. simulation-based training systems.

For the scenarios studied in this chapter, the greatest challenges were observed for the Risk-Aware Flight Route Selection scenario, where the tunable actor needs to select a flight route in continuous space. Finding a suitable route for every utility function in the utility space would require a large number of episodes of learning, which can require expensive computations for complex scenarios. As a first step towards further improving the efficiency of learning with multiple utility functions, the next chapter proposes a model-based approach to multi-objective reinforcement learning in environments where users’ utility functions change over time.
In some settings, users’ utility functions change over time. For a synthetic agent to maximise performance for the new utility function, a new policy must be adopted. One approach for handling utility changes is to learn a policy set in advance, so that when a utility change occurs, a new policy can simply be selected from the set. However, this approach can waste computations, since some of the learned policies may never match a user’s utility. Instead, the learning of a policy can be deferred to the time the new utility function is encountered, but at the cost of longer adaptation time. To find a trade-off between computation costs and adaptation time, this chapter proposes a model-based actor-critic, which explores with diverse utility functions through imagined rollouts within a learned world model between interactions with the real environment. An experimental evaluation shows that by learning a model of the environment, the performance of the agent can be improved compared to model-free algorithms. This chapter is based on Paper VIII [69] and Paper IX [70].

7.1 Adapting Online to Changes in User Utility

Changes in users’ utility functions can be triggered in several ways. Some changes in utility functions are triggered by changes in the environment. For instance, a commuter that normally travels to work by subway to save money may be willing to spend extra money for switching to travelling by taxi when there are delays in the subway system, in order to arrive in time for an im-
important meeting. Other changes in utility functions can occur when decision-making agents encounter new users. For instance, a self-driving taxi could allow each passenger to define their preferences for the journey, e.g., their preferences for travelling along a scenic route or arriving at their destination as quickly as possible. Similarly, in adaptive simulation-based training, the behaviour of the simulation should be adjusted as trainees improve their proficiency, so that they still receive stimulation when interacting with the training system. Furthermore, in each training session the contents of simulation scenarios should be adapted to the current needs of the participating trainees. In addition, changes in the operational environment may require that the behaviour of synthetic agents in the simulation must also change.

In MORL research, dealing with settings where users’ utility functions change is referred to as the D\textit{ynamic Utility Function Scenario} \[45\] (see Chapter 5). To handle changes in utility efficiently, it is desirable to reuse information from learning with previously encountered utility functions, instead of restarting learning from scratch. Instead of learning a full coverage set a priori, learning is focused on the utility functions encountered while the system is in operational use. As more and more users and utility functions are encountered, the learned coverage set will more closely approximate the full set of optimal policies. Then, when preferences over objectives change, the user can select a suitable policy from the policy set to use and improve upon.

Previous work in MORL for learning with dynamic utility functions has focused on model-free learning, which often suffers from poor sample efficiency. This is problematic in environments where only limited interaction with the real environment is available, for instance due to the cost of running experiments. Recently, model-based reinforcement learning methods have been improved, and demonstrated impressive performance on complicated tasks \[118, 42, 44\]. We believe that creating a model while learning could be particularly useful for multi-objective reinforcement learning, where many different solutions need to be explored within a single environment to find one that fits the user’s utility.

In this chapter, we therefore propose a model-based actor-critic (MO-Dreamer), based on DreamerV2 \[44\], as illustrated in Figure 7.1. To stabilise learning in the dynamic utility scenario we use Prioritised Diverse Experience Replay to ensure that the replay buffer contains trajectories with diverse multi-objective returns, and that diverse trajectories are sampled in the initial stages of learning. We then let the agent explore with diverse utility functions through imagined rollouts within the learned world model. An experimental evaluation on the Minecart benchmark \[3\] shows that the model-based agent significantly outperforms the model-free state-of-the-art for frequent as well as sparse utility changes. In additional experiments on the simpler Deep Sea Treasure benchmark \[142\], the model-based agent outperforms the model-free agents overall by converging quickly, but learns a worse final policy on average. When Deep Sea Treasure is configured with partial observability to
7.2 Related Work

This section presents work related to model-free multi-objective reinforcement learning in the dynamic utility function scenario, followed by an overview of model-based reinforcement learning.

Learning with Dynamic Utility Functions

Natarajan & Tadepalli proposed introducing multi-objective reinforcement learning with dynamic preferences in tabular settings. Their method learns and stores un-dominated policies for encountered preference weights. When a weight makes it more challenging, the model-based agent again provides a significant performance boost compared to the model-free baselines. To the best of our knowledge this is the first study of model-based multi-objective reinforcement learning in the dynamic utility function scenario. In the context of user-adaptive simulation-based training, it is a first step towards realising the world model component in the system architecture proposed in Chapter 5.

The remainder of this chapter is organised as follows. Section 7.2 presents related work on learning in the dynamic utility function scenario and model-based reinforcement learning. Section 7.3 proceeds to present the components of the MO-Dreamer agent, followed by an experimental evaluation in Section 7.4. Finally, section 7.5 provides a summary and directions for future work.

Figure 7.1: MO-Dreamer interacts with the environment and builds a dataset of diverse experiences, which are used to construct a model for imagination rollouts where past experienced states are revisited with experienced as well as imagined utility functions for improvement of the policy.

Figure 7.1: MO-Dreamer interacts with the environment and builds a dataset of diverse experiences, which are used to construct a model for imagination rollouts where past experienced states are revisited with experienced as well as imagined utility functions for improvement of the policy.

Figure 7.1: MO-Dreamer interacts with the environment and builds a dataset of diverse experiences, which are used to construct a model for imagination rollouts where past experienced states are revisited with experienced as well as imagined utility functions for improvement of the policy.
change occurs, learning continues with the past policy that has the highest scalarised value for the new weight vector: $V^* \cdot \mathbf{w}$. This is shown to improve performance compared to learning a new policy from scratch.

Abels et al. \cite{abels2017} extended learning with dynamic weights to multi-objective deep reinforcement learning. Instead of learning a set of individual policies, a single deep Q network (DQN \cite{mnih2015}) was used to represent multiple policies, by conditioning the network on the preference weights of objectives. This allows the network to generalise across weight changes. In addition, a mechanism for enforcing diversity in the replay buffer was proposed, to prevent the agent from forgetting policies learned in the past. Diversity is measured as the crowding distance \cite{lee2013} of the returns of stored trajectories, and when the buffer is full trajectories that contribute the least to diversity are removed first.

Nian et al. \cite{nie2020} extended the work of Abels et al. \cite{abels2017} to partially observable environments. They proposed a Deep Conditioned Actor-Critic (DCRAC), which can be equipped with LSTM or memory networks to form beliefs about the current state of the environment. The approach was evaluated on configurations of the Deep Sea Treasure benchmark with partial observability.

Wang et al. \cite{wang2020} proposed a new Near on-policy Experience Replay (NER) algorithm for settings where preference weights change rapidly, and the replay buffer may contain a large number of transitions that are not relevant for the current weight vector. This can result in large extrapolation errors \cite{poon2015}. The proposed method overcomes this problem by prioritising sampling of transitions that have states and weight vectors that are similar to the current state and weight vector.

Some model-based approaches have been proposed for MORL, e.g. variations of Monte Carlo tree search \cite{silver2016, silver2017}, but to the best of our knowledge they have not been studied in the dynamic utility setting. We extend prior work on learning with dynamic utility functions by studying the potential benefits of online learning of a world model that can be used to train the agent.

Model-Based Reinforcement Learning

In many real-world problems, the number of interactions that a learning agent can have with the environment are limited. For example, environments populated by humans may have low availability, and environments populated by expensive vehicles may have a high operating cost. In such settings sample efficiency can be improved by learning a model of the environment dynamics, which can then support decision making by running simulations.

Recent advances in model learning have made it possible to build accurate compact representations even for complex environments with image observations. The Dreamer agent \cite{andrychowicz2018} uses a world model consisting of three components: a representation model that provides compact vector-valued state representations of image observations, a transition model that predicts the
next model state based on the current model state and action, and a reward model that predicts the reward for the current model state. The agent learns the latent dynamics of the world model by using reconstruction of the images that represent the observations of the agent as a learning objective. This results in an efficient model that can simulate several thousands of trajectories in parallel. The agent achieves impressive results on challenging visual control tasks in DeepMind Control Suite \[132\].

DreamerV2 \[44\] provides an evolution of the Dreamer agent, which significantly improves the performance on the challenging Atari benchmark \[13\] of environments with discrete action spaces and image-based observations. DreamerV2 uses categorical variables to represent the latent state of the world model, instead of the diagonal Gaussian distribution used by the original Dreamer agent. In addition, KL (Kullback–Leibler) balancing is used to improve robustness to novel inputs as well as learning of long term dependencies in the environment. We use DreamerV2 as the basis for our work, and extend it to support learning in MOMDPs with dynamic utility functions.

### 7.3 Method

In this section we present the structure of the model-based multi-objective actor-critic, MO-Dreamer. First, the world model is presented, followed by a description of model learning by prioritised diverse experience replay. We then present the actor-critic, followed by a description of policy learning through imagination rollouts. We focus on the differences between MO-Dreamer and DreamerV2. For a detailed description of DreamerV2 we refer the reader to \[44\]. An overview of MO-Dreamer is shown in Figure 7.1 and pseudo code is given in Algorithm 2.

#### Multi-Objective World Model

We build upon the recurrent state space model (RSSM) proposed in DreamerV2 \[44\]. The RSSM consists of the recurrent model, representation model, and transition predictor, as shown in Equations (7.1) – (7.3), with $\phi$ representing the parameter vector of the neural networks of the world model.

\[
h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}), \tag{7.1}
\]

\[
z_t \sim q_\phi(z_t|h_t, x_t), \tag{7.2}
\]

\[
\hat{z}_t \sim p_\phi(\hat{z}_t|h_t), \tag{7.3}
\]
7. Dreaming with Dynamic Utility Functions

Algorithm 2 MO-Dreamer

Initialise neural network parameters randomly.
Prefill replay buffer with exploration policy.

while $t < t_{\text{max}}$ do
  for $c ← 1$ to $N_{\text{train}}$ do
    $p_{\text{main}} = N_{\text{main}} / N_{\text{tot}}$ \quad $\triangleright$ probability of sampling main buffer
    $p_{\text{secondary}} = 1 - p_{\text{main}}$ \quad $\triangleright$ probability of sampling secondary buffer
    Sample batch $B$ with chunks from replay buffers.
    Train world model with $B$.
    Select mix of current, past and imagined weights $\{w_i\}$.
    Imagine trajectories $\{(z_\tau, a_\tau, w_i)\}_{\tau=t}^{t+H}$ from each initial model state-utility pair $(z_t, w_i)$.
    Predict rewards $\hat{r}_t \sim p_\phi(\hat{r}_t|h_t, z_t)$ and values.
    Train actor-critic with data from imagination rollouts.
  for $\tau ← t$ to $t + T$ do
    Compute action $\hat{a}_\tau \sim p_\psi(\hat{a}_\tau|\hat{z}_\tau, w)$
    Step environment $o_{\tau+1}, r_{\tau+1}, \text{done} ← \text{env.step}(\hat{a}_\tau)$
    if done then $o_{\tau+1} ← \text{env.reset}()$
    Add new experience to main replay buffer.
    Enforce sample limit and diversity in main buffer.
    Enforce sample limit in secondary buffer.

The RSSM uses the deterministic recurrent states $h_t$ to compute distributions over the posterior state $z_t$ and the prior state $\hat{z}_t$. The posterior state incorporates information about the image input $x_t$, and the prior state tries to predict the posterior state without access to $x_t$. The model state captures the current environment state based on observing a sequence of images of past environment states, and the model is able to predict forward in time to support "imagination" rollouts.

We also use the same predictors as DreamerV2 for state images, rewards, and discount (which is used to estimate the end of episodes), except that the reward is now a vector:

$$\hat{x}_t \sim p_\phi(\hat{x}_t|h_t, z_t), \quad (7.4)$$

$$\hat{r}_t \sim p_\phi(\hat{r}_t|h_t, z_t), \quad (7.5)$$

$$\hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t|h_t, z_t), \quad (7.6)$$
7.3. Method

As in DreamerV2, the latent state of the model is represented with categorical variables, the image predictor is represented by a diagonal Gaussian with unit variance, and the discount predictor is represented by a Bernoulli likelihood. We make the assumption that the elements of the multi-objective reward are statistically independent, and represent them as individual univariate Gaussians with unit variance in the world model. Then the updated loss function for a world model of an \( n \)-objective MOMDP is:

\[
L(\phi) \doteq E_{q_\phi(z_{1:T}|h_{1:T},x_{1:T})} \left[ \sum_{t=1}^{T} -\ln p_\phi(x_t|h_t,z_t) - \sum_{i=1}^{n} \ln p_\phi(r_i|h_t,z_t) - \ln p_\phi(\gamma_t|h_t,z_t) + \beta KL[q_\phi(z_t|h_t,x_t)||p_\phi(z_t|h_t)] \right].
\] (7.7)

The loss term for KL balancing controls the difference between the distributions of prior \((\hat{z}_t)\) and posterior \((z_t)\) stochastic states.

Prioritised Diverse Experience Replay

The world model is trained with data collected from the agent’s past experiences with the real environment, which have been stored in a replay buffer. In addition to sequences of image observations, actions, rewards, and discount factors, the stored experiences also contain information about the utility weights that were active in the corresponding episode. DreamerV2 randomly samples trajectories from the buffer to construct batches of sequences with fixed length. In the dynamic utility scenario it is necessary for the agent to learn that different policies should be followed for different utility functions, and to retain that knowledge over time. Abels et al. [3] addressed this issue by splitting the replay buffer of a DQN agent into a main buffer and a secondary buffer that enforces diversity in terms of the multi-objective returns contained in the buffer. When the buffer becomes full and samples must be removed, the trajectories that contribute the least to diversity are removed first. We apply a variation of this technique to the replay buffer of MO-Dreamer.

When using model-based learning in the dynamic utility scenario, it is important to quickly learn a sufficiently accurate model of the achievable rewards in different parts of the environment. To promote learning environment features suitable for multiple utility functions we want to ensure diversity in the data used in the early stages of learning. We therefore use two replay buffers, as in [3], but enforce diversity on the main buffer. The diversity mechanism thereby comes into play as soon as the main buffer is filled (while in [3] it does not come into play until both buffers are filled). When the main buffer is full, the trajectory that contributes the least to diversity is moved to the secondary buffer, which uses a first-in-first-out (FIFO) principle. We then sample trajectories from either buffer with a probability proportional to...
7. Dreaming with Dynamic Utility Functions

the number of samples contained in each of them. This means that the early stages of learning will prioritise sampling trajectories with diverse outcome in terms of the multi-objective return, while in later stages of learning the diversity buffer and secondary buffer will be sampled with equal probability.

**Actor-Critic for Dynamic Utility**

We use an actor-critic setup to learn the behaviour of the agent, where the critic learns a multi-objective value function that guides the updates of the actor’s policy. To enable single-network representations of the action distributions as well as the state value functions of multiple policies, each corresponding to a different utility function, we condition both actor and critic on the current utility weights:

$$\hat{a}_t \sim p_\psi(\hat{a}_t|\hat{z}_t, w),$$  \hspace{1cm} (7.8)

$$v_\xi(\hat{z}_t, w) \approx E_{p_\phi} \left[ \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau | w \right],$$  \hspace{1cm} (7.9)

where $\psi$ and $\xi$ are the parameters of the actor and critic respectively. When the critic gets a certain weight vector as input in combination with the current model state, it will output the corresponding vector of optimal objective values from its current estimation of the CCS. When the actor gets a certain weight vector as input, it will select the best policy for optimising the corresponding utility.

To learn the value function of the multi-objective critic we use temporal-difference learning with the vector $\lambda$-return as target value:

$$V^\lambda_t \doteq \hat{r}_t + \gamma_t \left\{ (1-\lambda)v_\xi(\hat{z}_{t+1}, w) + \lambda V^\lambda_{t+1} \right\} \text{ if } t < H,$$

$$v_\xi(\hat{z}_H, w) \text{ if } t = H,$$

(7.10)

and optimise using the squared loss:

$$\mathcal{L}(\xi) \doteq E_{p_\phi} \left[ \sum_{t=1}^{H-1} \frac{1}{2} \| v_\xi(\hat{z}_t, w) - s_g(V^\lambda_t) \|^2 \right],$$  \hspace{1cm} (7.11)

where $s_g$ refers to stopped gradients.

To learn the policy of the actor we use the scalarised advantage function as a baseline and optimise with the following Reinforce loss:

$$\mathcal{L}(\xi) \doteq E_{p_\phi} \left[ \sum_{t=1}^{H-1} \frac{1}{2} \| v_\xi(\hat{z}_t, w) - s_g(V^\lambda_t) \|^2 \right],$$  \hspace{1cm} (7.11)
7.3. Method

\[ L(\xi) \triangleq E_{p_\psi, p_\psi} \left[ \sum_{t=1}^{H-1} \left( -\ln p_\psi(\alpha_t|\hat{z}_t, w) \cdot \text{sgn}((V^\lambda_t - v_\xi(\hat{z}_t, w)) \cdot w) \right) \right] - \eta H[\alpha_t|\hat{z}_t, w] \]  

(7.12)

where the weight vector \( w \) is the utility function that was active when the corresponding samples of a batch was collected. The entropy term \( H \) in the actor loss can help prevent over-fitting, which is of particular importance in the dynamic utility scenario, where the agent needs to do transfer learning between different preference weights. The parameter \( \eta \) controls the importance of entropy in the total actor loss.

Since the actor-critic is learning from the compact hidden state of the world model, the actor and critic are represented by simple MLPs, rather than the CNNs that would typically be used when learning directly from image input.

### Imagination Rollouts

The MO-Dreamer actor-critic is trained through "imagination" rollouts in the learned world model. The initial state of each rollout corresponds to a time step of a sequence sampled from the replay buffer and then encoded to a compact world model state. In addition to the compact representation of the experienced real environment state, the agent is also provided with the utility function that was active in the episode where the state was experienced. Either this utility function or the current utility function of the environment are used by the agent when doing rollouts in the world model and optimising the policy with the sample batch collected. Continually revisiting past trajectories prevents the agent from forgetting which policy to use for utility functions encountered in the past, and the robustness of the agent’s behaviour is improved. The diverse experience replay mechanism ensures that there remains a diverse mixture of initial states and utility functions in the replay buffer.

In the initial stages of learning the replay buffer is filled with experiences gathered by an exploration policy, and the trajectories followed may not be at all optimal for the actual utility function of the episode. Since we are learning with a model, we are able to revisit states previously encountered by the agent, but with a different utility function, to explore which parts of the environment provide most value for that function. We implement this mechanism in the exploration phase of the agent, by performing additional imagination rollouts with imagined utility functions sampled by the agent, rather than the utility functions experienced in the past. This results in a diverse mixture of initial states and utility functions, which can help the agent figure out which parts of the environment to visit when given a utility function in the future.
7. Dreaming with Dynamic Utility Functions

Figure 7.2: From left to right, Minecart, Partially Observable Minecart, Deep Sea Treasure, and Partially Observable Deep Sea Treasure evaluation environments.

7.4 Experimental Evaluation

This section presents the experimental evaluation of MO-Dreamer. First, we describe the environments and evaluation metrics used. Second, we provide a summary of the algorithms studied and their settings. Finally, the results of the experiments are presented.

Experiment Setup

In this work, we study the dynamic utility function scenario with frequent and sparse utility changes. In air combat training, frequent utility changes can correspond to the training system interacting with and adapting to different users in each training session. Sparse changes in utility can correspond to the system adapting to changes in the training needs of an individual trainee. As in previous work, we use a linear utility function represented by objective weights, which are sampled from a Dirichlet distribution ($\alpha = 1$) when changes in utility functions occur. We perform experiments on the well-known MORL benchmarks Minecart [3] and Deep Sea Treasure [3, 142], illustrated in Figure 7.2. These environments were also used in prior work on learning with dynamic utility functions [3, 96, 151]. The design of the environments ensure that there are several distinct optimal policies, each of them optimal for some set of utility functions. This means that when utility functions change, the learning agents must consider switching to a new policy, to avoid sub-optimal performance.

In Minecart, illustrated in Figure 7.2, the agent operates a minecart (white icon) to different mines (black areas) for mining different ores, which can then be brought back to the home base (red area) to be sold. The current contents of the cart is displayed as colour bars within the cart icon. The available actions allow the agent to Mine, turn Left or Right, Accelerate, Brake, or Idle. The objectives are related to the values of the different ores once sold, and the fuel cost caused by operating the minecart. Episodes end when the minecart returns to the base, or when a maximum of 1000 time steps have passed. Minecart is one of the more challenging benchmarks for multi-
objective reinforcement learning, with a high-dimensional observation space in the form of an image of the environment state (as in Figure 7.2), stochastic state transitions when mining (based on the ore distribution specified for each mine), and delayed rewards for mining and selling ores. We use the default configuration of Minecart \cite{135}, which has two ores. In this environment we use a discount factor of $\gamma = 0.98$.

In Deep Sea Treasure the agent operates a submarine (white square) to treasures (green squares). Deeper treasures have higher value, but there is a penalty for each time step in the episode, resulting in two objectives. The available actions allow the agent to step Left, Right, Up or Down. Episodes end when a treasure is collected, or when a maximum of 1000 time steps have passed. Observations are given as images of the environment state (as in Figure 7.2). We use the configuration from \cite{135}, where the value of collecting each treasure lies in the CCS. In this configuration each time step gives a penalty of -1, and the values of treasures from left to right are $\{18, 26, 31, 44, 48.2, 56, 72, 76.3, 90, 100\}$. In this environment we use a discount factor of $\gamma = 0.95$.

We also run experiments that study the world model’s ability to handle partial observability, by only providing a 240x240 pixels observation centred on the agent instead of the 480x480 pixels observation of the full environment in Minecart, and by only providing a 5x5 cells observation centred on the agent instead of the 12x12 cells observation of the full environment in Deep Sea Treasure (as illustrated in Figure 7.2).

From a pilot training perspective, Minecart has some characteristics that could also be found in a simplified air combat simulation. For instance, the vehicle dynamics are similar to those of a rotary wing aircraft. The task of moving to mines to collect different types of ores is similar to the task of completing a surveillance mission, where the aircraft moves to points of interest to gather different types of intelligence information. Requiring agents to deal with partial observability is also important in air combat simulation, where limitations of sensors and effects of electronic warfare must be dealt with.

The main challenge of the Deep Sea Treasure environment is the trade-off between exploration and exploitation. This may result in the agent getting stuck at the treasures in the shallow part of the environment, and not finding policies for reaching the more valuable treasures in the deeper sea. The challenge of exploration is expected to be found in air combat scenarios as well, due to the complexity of aircraft systems and the environment aircraft operate in. For instance, in Chapter \cite{135} it was observed that finding routes with an appropriate risk-level can be challenging.

Ideally we would like to use regret as a metric to compare the results of different algorithms. The regret is defined as the difference between the optimal utility and the actual utility for a given utility function, $\Delta(g, w) = V^*_w \cdot w - g \cdot w$, where $V^*_w$ is the optimal value in the CCS for the current weight.
7. Dreaming with Dynamic Utility Functions

vector, and $g$ is the discounted return. The regret metric can consistently evaluate performance over different runs and for different weight vectors.

Since the optimal policy for Minecart is not known, we instead use the heuristic proposed by [3] to estimate an approximate CCS. Since we are using an estimate for the optimal utility, there is a chance that the regret estimate could become negative, if the RL agents learn policies that outperform the heuristic. To avoid this, we estimate the optimal episodic utility with the maximum of $V_w^* \cdot w$ and the highest utility achieved by any learning agent. A negative side effect of this metric is that we can only evaluate for the fewest episodes completed by any learning agent, which was found to be roughly 38,000 episodes for CN-PER (as defined in the algorithms overview in the next section) when running the experiments. For Deep Sea Treasure the optimal policy is known, so for experiments in that environment regret is evaluated for all learning steps. Experiments in Minecart last for 1 million time steps in the environment, experiments in Deep Sea Treasure last for 100,000 time steps, and each experiment is run for ten iterations to reduce the effects of random variations. The mean cumulative regret and the mean episodic regret are then evaluated.

Algorithms

In the experiments we compare MO-Dreamer to the existing state-of-the-art model-free algorithms for learning with dynamic utility functions presented in Section 7.2. We also evaluate MO-Dreamer to ablated versions of itself, to see how different components of the agent affects its performance. We study the following agents:

- **MO-Dreamer**: Full model-based agent with utility conditioned actor-critic networks, prioritised diverse experience replay, and imagined utility functions during exploration
- **MO-Dreamer-No-DER**: Ablation study of MO-Dreamer without prioritised diverse experience replay and sampling
- **MO-Dreamer-No-IU**: Ablation study of MO-Dreamer without imagined utility functions during exploration
- **MO-Dreamer-PO**: MO-Dreamer with partial observability
- **CN-NER/PER**: Model-free baseline with utility conditioned DQN and diverse experience replay according to [3], combined with either near on-policy experience sampling (NER) according to [151] or standard prioritised experience replay (PER) [117]
- **DCRAC-NER/PER**: Model-free baseline with utility conditioned recurrent actor-critic and diverse experience replay according to [96], com-
bined with either near on-policy experience sampling (NER) according to [151] or standard prioritised experience replay (PER) [117].

We configure MO-Dreamer to train every 10 steps in the real environment for Minecart, and every 5 steps for Deep Sea Treasure. Imagination rollouts have a horizon of 15 steps in the world model. We train with batches of 10 sequences of 50 steps, sampled from the replay buffer. The replay buffer is configured with a capacity of 200,000 steps for Minecart and 20,000 steps for Deep Sea Treasure. We prefill the buffer with 100,000 exploration steps for Minecart and 10,000 exploration steps for Deep Sea Treasure (corresponding to the size of the main buffer), before starting normal training. In Minecart we explore using a random policy. In Deep Sea Treasure we instead use model-based exploration with Plan2Explore [119], since random exploration is inefficient for finding the deeper treasures. Plan2Explore provides intrinsic motivation to the agent for exploring parts of the environment where the model quality is low.

Up until the first 20% environment steps we perform 8 training iterations between each interaction with the environment, to quickly learn a high quality world model and policy. Half of these iterations use imagined utility functions. After 20% steps we reduce the training intensity to 2 iterations between each interaction with the environment (primarily to reduce the amount of wall time required to run the experiments), and only use current and past experienced utility functions in rollouts to focus learning on experienced combinations of state and utility function.

The reference implementation of DreamerV2 [44] is used as a basis for the implementation of MO-Dreamer. In experiments we use the default settings of DreamerV2 for the actor and critic networks. For the model networks we use the default configuration in Minecart, while in Deep Sea Treasure we use the simpler ATARI configuration, since Deep Sea Treasure has a simpler observation space. The Adam optimiser [76] is used for training, with learning rates of $\alpha = 2 \times 10^{-4}$ for the model and critic, and $\alpha = 8 \times 10^{-5}$ for the actor.

For the model-free agents we use the reference implementation and hyperparameters of CN-PER provided by [3] as a basis, and then add the near on-policy sampling mechanism proposed by [151] with its default hyperparameters.

**Results**

The average cumulative regret and standard deviation over ten runs on Minecart is shown in Figure 7.3. It can be seen that the model-based agents need fewer steps to complete the same number of episodes as the model-free agents. It can also be seen that MO-Dreamer outperforms the model-free baselines in terms of average cumulative regret, for frequent as well as sparse utility changes, indicating improved sample-efficiency. The algorithm con-
7. Dreaming with Dynamic Utility Functions

verges quickly after the exploration phase has ended at 100,000 environment steps, which is when the main replay buffer is filled. The two model-free baselines almost overlap in both settings, although they accumulate different amounts of regret after completing all episodes of learning. In our experiments, CN-PER performs better with frequently changing utility functions than what was reported by [151], but it is outperformed by CN-NER. For sparse utility changes, CN-PER has slightly better performance on average.

MO-Dreamer without diverse experience replay performs worse than the full algorithm, especially so when learning with sparse utility changes. This is likely caused by over-fitting. Qualitatively we could observe during training that in some runs an agent would have a bias for one of the ore’s mined, regardless of the current utility function. The agent would often recover from this sub-optimal behaviour later in training, but at that point a lot of regret could have been accumulated. The recovery from sub-optimal behaviour indicates that it is beneficial for a learning agent to be able to revisit a previously visited area of the environment and improve upon its behaviour for a given utility function.

When learning with sparse utility changes, using imagined utility functions is essential for good performance, as illustrated by Figure 7.3. This mechanism allows the agent to prepare for dealing with utility functions not yet encountered in the real environment. When learning with frequently changing utility functions the mechanism is less important, since the replay buffer will still contain a sufficiently large set of diverse examples of preference weights.

Providing MO-Dreamer with only partial observations of the environment instead of full observations only has a small negative impact on the agent’s performance. For frequent utility changes MO-Dreamer-PO’s regret curve almost overlaps the regret curve of MO-Dreamer-No-DER up until 500,000 steps, where MO-Dreamer-PO starts performing better. For sparse utility changes MO-Dreamer-PO’s performance is close to that of the full MO-
7.4. Experimental Evaluation

Table 7.1: Average episodic regret on Minecart for sparse and frequent utility changes.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Δ overall</th>
<th>Δ after 200,000 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent Utility Changes (every episode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN-NER</td>
<td>0.1215 ± 0.0057</td>
<td>0.0959 ± 0.0053</td>
</tr>
<tr>
<td>CN-PER</td>
<td>0.1278 ± 0.0097</td>
<td>0.1030 ± 0.0089</td>
</tr>
<tr>
<td>MO-Dreamer</td>
<td><strong>0.0661 ± 0.0226</strong></td>
<td><strong>0.0377 ± 0.0228</strong></td>
</tr>
<tr>
<td>MO-Dreamer-No-DER</td>
<td>0.0786 ± 0.0342</td>
<td>0.0529 ± 0.0380</td>
</tr>
<tr>
<td>MO-Dreamer-No-IU</td>
<td>0.0682 ± 0.0132</td>
<td>0.0389 ± 0.0120</td>
</tr>
<tr>
<td>MO-Dreamer-PO</td>
<td><strong>0.0716 ± 0.0257</strong></td>
<td><strong>0.0466 ± 0.0260</strong></td>
</tr>
<tr>
<td>Sparse Utility Changes (every 1000 episodes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN-NER</td>
<td>0.1313 ± 0.0131</td>
<td>0.1079 ± 0.0142</td>
</tr>
<tr>
<td>CN-PER</td>
<td>0.1291 ± 0.0199</td>
<td>0.1043 ± 0.0217</td>
</tr>
<tr>
<td>MO-Dreamer</td>
<td><strong>0.0605 ± 0.0130</strong></td>
<td><strong>0.0309 ± 0.0081</strong></td>
</tr>
<tr>
<td>MO-Dreamer-No-DER</td>
<td>0.1011 ± 0.0381</td>
<td>0.0753 ± 0.0431</td>
</tr>
<tr>
<td>MO-Dreamer-No-IU</td>
<td>0.0919 ± 0.0314</td>
<td>0.0476 ± 0.0179</td>
</tr>
<tr>
<td>MO-Dreamer-PO</td>
<td><strong>0.0704 ± 0.0186</strong></td>
<td><strong>0.0414 ± 0.0146</strong></td>
</tr>
</tbody>
</table>

Dreamer agent, while outperforming the other model-based agents. In both settings MO-Dreamer-PO outperforms the model-free baselines, even though they are learning with full observability.

Figure 7.3 shows that there is more variance in the results for MO-Dreamer compared to the model-free baselines. The variance is highly affected by outliers in the results. This experiment uses a high training intensity in the early stages of learning. There is a risk that an agent will learn sub-optimal policies during the period when all relevant features are not available in the compact state of the world model, e.g., idling at the home base to minimise fuel costs instead of moving to a mine to collect ores. It was noted when running the experiments that the representation of the cart’s contents took time to learn. This problem of “vanishing objects” has also been noted in previous work [98]. In the early stages of learning this issue could, e.g., result in the learning agent being rewarded for bringing back ores to the home base, even though it cannot observe that there are actual ores stored in the cart.

Table 7.1 presents the average episodic regret overall, as well as after 200,000 environment steps for Minecart. MO-Dreamer significantly outperforms the model-free baselines for frequent as well as sparse utility changes.

The average cumulative regret and standard deviation over ten runs on Deep Sea Treasure is shown in Figure 7.4. Since the optimal policy is known in this environment, we evaluate all agents over the full 100,000 time steps of learning. MO-Dreamer accumulates less regret than the model-free baselines.
for frequent as well as sparse utility changes, with full as well as partial observability. However, compared to Minecart the comparison of agents in the fully observable setting is more affected by noise in the results. In a typical run of training MO-Dreamer with 10 iterations, there were a few outliers that performed significantly worse than the other iterations, and highly affected the mean and standard deviation of the experiment. This is similar to the behaviour related to overfitting that was also observed on Minecart, but more severe. After convergence model-free agents with near on-policy experience replay have learned a better policy than MO-Dreamer for frequent as well as sparse utility changes, while model-free agents with standard prioritised experience replay on average learn better final policies than MO-Dreamer for frequent changes in utility, but not for sparse changes.

One possible reason for outliers and worse final performance of MO-Dreamer compared to model-free agents is likely a worse exploration policy, which results in fewer episodes that reach the deeper treasures. It is also possible that the replay buffer is too small, resulting in only a limited number of trajectories that reach the deeper rewards being stored in the buffer. In addition, the batches used for training of MO-Dreamer are assembled from sampled chunks of the stored trajectories, which means that for long tra-
Table 7.2: Average episodic regret on Deep Sea Treasure for sparse and frequent utility changes.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>∆ overall</th>
<th>∆ after 20,000 steps</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequent Utility Changes (every episode)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN-NER</td>
<td>0.7885 ± 0.1616</td>
<td>0.3428 ± 0.0949</td>
</tr>
<tr>
<td>CN-PER</td>
<td>0.7705 ± 0.1629</td>
<td><strong>0.3295 ± 0.1010</strong></td>
</tr>
<tr>
<td>DCRAC-NER</td>
<td>4.3620 ± 0.6683</td>
<td>2.6365 ± 1.1497</td>
</tr>
<tr>
<td>DCRAC-PER</td>
<td>4.6200 ± 0.6903</td>
<td>3.0395 ± 1.1180</td>
</tr>
<tr>
<td>MO-Dreamer</td>
<td><strong>0.6498 ± 0.1428</strong></td>
<td>0.4585 ± 0.1677</td>
</tr>
<tr>
<td>MO-Dreamer-No-DER</td>
<td>0.8516 ± 0.3551</td>
<td>0.7259 ± 0.3851</td>
</tr>
<tr>
<td>MO-Dreamer-No-IU</td>
<td>0.9172 ± 0.4554</td>
<td>0.5994 ± 0.4201</td>
</tr>
<tr>
<td>MO-Dreamer-PO</td>
<td><strong>1.2968 ± 0.5296</strong></td>
<td><strong>1.1965 ± 0.5358</strong></td>
</tr>
<tr>
<td><strong>Sparse Utility Changes (every 100 episodes)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CN-NER</td>
<td>0.8429 ± 0.1949</td>
<td><strong>0.3385 ± 0.0854</strong></td>
</tr>
<tr>
<td>CN-PER</td>
<td>0.9092 ± 0.2745</td>
<td>0.4123 ± 0.2138</td>
</tr>
<tr>
<td>DCRAC-NER</td>
<td>4.0633 ± 0.2754</td>
<td>1.9793 ± 0.4694</td>
</tr>
<tr>
<td>DCRAC-PER</td>
<td>4.7159 ± 1.2038</td>
<td>3.1187 ± 1.8380</td>
</tr>
<tr>
<td>MO-Dreamer</td>
<td><strong>0.7900 ± 0.3808</strong></td>
<td>0.3822 ± 0.2136</td>
</tr>
<tr>
<td>MO-Dreamer-No-DER</td>
<td>0.8293 ± 0.2781</td>
<td>0.6535 ± 0.3339</td>
</tr>
<tr>
<td>MO-Dreamer-No-IU</td>
<td>1.1690 ± 0.4647</td>
<td>0.5756 ± 0.3617</td>
</tr>
<tr>
<td>MO-Dreamer-PO</td>
<td><strong>1.4492 ± 0.9789</strong></td>
<td><strong>1.3302 ± 1.0172</strong></td>
</tr>
</tbody>
</table>

Table 7.2 presents the average episodic regret overall, as well as after 20,000 environment steps for Deep Sea Treasure with full and partial observability. When learning with full observability MO-Dreamer has the best performance overall, but the model-free agents produce the best results for the final 80,000 steps. For Partially Observable Deep Sea Treasure MO-Dreamer significantly outperforms the model-free baselines overall. For the final 80,000 steps, MO-Dreamer significantly outperforms the model-free baselines for frequent utility changes, while for sparse utility changes MO-Dreamer has the best performance on average, but with higher variance than DCRAC-NER.

Finally, we performed a qualitative evaluation of MO-Dreamer. We trained the agent for 1 million steps in the deterministic configuration of Minecart, with frequent utility changes (every episode). The agent’s behaviour was then studied for different utility functions. We could see that MO-Dreamer trajectories the chunk may not contain the episode end, where the treasure is collected. Increasing the buffer size and using an improved sampling mechanism could improve performance. Other ways of improving performance could be to adjust the entropy coefficient, reduce the model capacity, or enhance the exploration policy.
learns precise navigation in the environment. An example episode is given in Figure 7.5, where the agent’s observations for each time step are presented from left to right and top to bottom of the figure. For this episode the utility weights were \( w = [9.9 \times 10^{-1}, 7.7 \times 10^{-3}, 8.4 \times 10^{-5}] \), i.e. there is a strong focus on mining and selling the first ore. Since we are using a discount factor \( \gamma < 1 \) and the reward for mining the ore is delayed until reaching the home base, it is important to complete the episode with as few steps as possible to maximise the discounted return. In the context of air combat simulation, and our reinterpretation of Minecart as a surveillance mission, the discount factor could be used to model that the value of intelligence information is reduced if it arrives too late at a command and control centre. The figure shows that MO-Dreamer chooses to accelerate fast straight from the home base, and then performs a U-turn to arrive at the target mine facing the home base. Then ores are mined before returning to the base to receive the reward. In contrast, the heuristic proposed for Minecart first rotates the cart to face the target mine, while being stationary at the home base. Then the cart drives to the target mine and stops. After filling the cart with ores, the cart is rotated to face the home base, once again while being stationary at the mine. Then the cart is accelerated to return home. For the studied episode, MO-Dreamer manages to complete the episode in two time steps fewer, which results in MO-Dreamer achieving 4.1% higher utility than the man-made heuristic. How to navigate in general depends on the utility function, since the fuel cost must be considered in relation to the values of the ores. We could observe that MO-Dreamer’s way of navigating changed based on the current utility function, even when selecting the same target mine.
7.5 Summary

In this chapter, we proposed MO-Dreamer, a model-based multi-objective actor-critic for learning in environments with dynamic utility functions. MO-Dreamer enforces diversity in the returns of the trajectories stored in and sampled from the experience replay buffer, to enable high-intensity training early in the learning process with reduced risk of over-fitting. In addition, MO-Dreamer uses imagination rollouts with a diverse set of utility functions, to explore which policy to follow to optimise the return for a given set of objective preferences.

An experimental evaluation on the Minecart benchmark with frequent as well as sparse changes in utility functions showed that MO-Dreamer significantly outperforms the model-free state-of-the-art algorithms for multi-objective reinforcement learning in the dynamic utility function scenario in terms of cumulative regret and average episodic regret. On the Deep Sea Treasure benchmark, MO-Dreamer outperforms the model-free agents overall by converging quickly, but learns a worse final policy for full observability. With partial observability, MO-Dreamer significantly outperforms the model-free baselines. The fact that MO-Dreamer can perform competitively with model-free algorithms on the simple Deep Sea Treasure task illustrates the efficiency of the world model learning.

The model-based approach makes it possible to adapt more efficiently to changes in trainees training needs. In an operational training system, the world model could be optimised based on data from each training session. Then, the model can be used to optimise policies for synthetic agents that are optimal for every utility function observed for the population of trainees in a training centre. Compared to the simulation provided by the tactical environment simulation of a flight simulator, the learned world model is much more compact and compute-efficient. This means that learning with the world model allows policies to be fine-tuned to a higher degree, provided that the quality of the model is good enough.

In future work we intend to extend the world model to handle environments with multiple learning agents. In addition to supporting learning of policies, such a world model could be used online, to make predictions about other agent’s future decision-making. This could support decision-making in cooperative as well as competitive scenarios. Another interesting direction for future work is to study how the learned world model can be used for various forms of transfer learning. For instance, we would like to study how the world model learned when acting with a linear utility function can be used to transfer to non-linear utility functions. This might require new exploration strategies, to improve the world model in parts of the environment that are relevant for non-linear utility functions but not for linear ones. Finally, we would like to extend the experimental evaluations to environments that are more relevant for tactical pilot training.
The aim of this dissertation was to analyse and model simulation-based training systems incorporating synthetic learning agents for improvement of efficiency and effectiveness; and to support delivery of adaptive simulation-based training by developing methods for utility-based reinforcement learning, which aims to provide an optimal solution based on knowledge about the application domain and the utility functions of its users. The long-term goal is to enable efficient adaptation of training to the needs of each individual trainee, instead of providing a one-size-fits-all solution. This chapter first presents a summary of our contributions in Sections 8.1 and 8.2, and then continues to discuss suitable directions for future research in Section 8.3. The chapter and dissertation is then concluded with a few final words.

8.1 Summary of Contributions

In Chapter 1, the following research questions were posed:

RQ1: What capabilities and characteristics do synthetic agents need to participate as actors in mixed cooperative and competitive simulation-based training scenarios?

RQ2: How can reinforcement learning methods assist an instructor in delivering training?
RQ3: How can simulation contents be adapted to fit the training needs of an individual trainee?

In part I of the dissertation we analysed how agent technologies relate to constraints imposed on actors in training systems, and what decision-making patterns should be supported by agent designs. Through interviews, a survey, and practical experiments we learned about requirements on agent capabilities and characteristics, challenges and shortcomings of current agent technologies, and aspects of human-agent interaction with agents constructed using state of the art reinforcement learning techniques. Finally, we discussed design approaches and solution concepts for a user-adaptive training system architecture that integrates learning agents.

The user and domain analysis contributes towards RQ1 by improving our understanding of the user in different types of training. One of the most important abilities identified is adaptability in various forms. When moving from basic training through tactical procedure training to mission training, synthetic agents should adapt and become more challenging opponents. They may also be assigned new roles, moving from the role of group leader to more frequently act in a supporting role to a more proficient human pilot. Adaptability was also identified as a desirable characteristic of learning algorithms, so that more or less domain knowledge in the form of doctrinal behaviour can be embedded in the synthetic agents. The practical experiments illustrate the importance of using high-level actions with a temporal extension to improve learning efficiency, or alternatively using a training scheme such as curriculum learning to improve agent’s performance by having them face successively greater challenges.

The system concept provides a structure to guide future research related to RQ2 and RQ3, with two major categories of synthetic, learning agents to support training: the Scenario Adaptation Agent and the Synthetic Trainer Agent. We propose to construct these agents using multi-objective reinforcement learning, so that they can deal with learning objectives as well as the objectives of air combat scenarios. Compared to single-objective reinforcement learning, multi-objective reinforcement learning provides greater flexibility in modelling human behaviour, e.g. by using non-linear utility functions. Multi-objective reinforcement can also improve sample-efficiency when multiple policies must be learned, and makes agent behaviour more transparent by providing vector value functions.

In part II of the dissertation, two agents for multi-objective reinforcement learning were proposed: the Tunable Actor (T-Actor) and the Multi-Objective Dreamer (MO-Dreamer). T-Actor addresses the Decision Support scenario of multi-objective reinforcement learning, by learning an approximate convex coverage set that can be presented to an instructor. By studying agent behaviour for different utility functions, the instructor can find a suitable agent (or set of agents) to use in a training session. This contributes towards re-
search questions RQ2 and RQ3. MO-Dreamer addresses the Dynamic Utility Function scenario of multi-objective reinforcement learning, by learning a model of the environment that helps it adapt online to changes in user utility. Further extensions of this method could lead to agents that can adapt to user needs online in training sessions, in a similar way as human role-players, thereby contributing towards RQ3. This would also result in an automation of part of the instructor’s work, thereby contributing towards RQ2.

8.2 Expected Impact

Overall, the research presented in this dissertation contributes to improved knowledge about how to apply machine learning methods to construction of simulation-based training environments. The work forms a basis for improvement of air combat training. Our proposed architecture for a user-adaptive training system, as well as our two utility-based agents, illustrate the potential advantage of moving from the predominant approach of single-objective learning with scalar rewards, and instead use the more general approach of utility-based multi-objective reinforcement learning. While the focus in this dissertation was on air combat training as an example of simulation-based training, many of the issues and techniques studied are general enough to be applicable in other domains. For instance, the need to evaluate trainees and adapt their training environment to improve their proficiency as quickly as possible. A similar approach could be taken in training of pilots for civilian flight, where agents could help realise dense air traffic and patterns of life, automated setting of adversarial weather conditions and malfunctions, as well as automated evaluation and profiling of trainees. What at first might stand out the most in the air combat domain compared to other domains is the competitive aspects of the environment. However, this aspect can also be found in other domains related to public safety, for instance in training of police officers or security guards. In addition, a competitive element can also be found in training systems that adopt gamification as a method to inspire users to perform well.

In addition to the contributions in the domain of simulation-based training, the two proposed utility-based agents also represent a general contribution to the field of multi-objective reinforcement learning. In particular, they contribute to improved efficiency in settings where multiple policies must be learned to optimise users’ utility. The benchmark problems studied in multi-objective reinforcement learning have been quite simple compared to real-world problems where multi-objective decision-making is needed [15]. As researchers try to tackle more challenging problems, the efficiency of learning algorithms will likely be of high importance.
8.3 Future Work

This section discusses important research directions for extending the work presented in this dissertation.

Design of Utility Functions and Reward Signals

The multi-objective learning studied in this dissertation has used linear utility functions, which are quite limited. In air combat training, it is expected that non-linear utility functions would be useful for modelling the decision-making of a synthetic pilot, as well as the learning objectives of trainees. For instance, utility functions that use thresholds could be used to model risk-aware decision-making as well as sufficient proficiency in a learning objective. When non-linear utility functions are used, it becomes necessary to choose if the optimisation should be based on the scalarised expected returns (SER), or the expected scalarised returns (ESR). For a user-adaptive training system, we argue that ESR is the correct optimisation criterion, since we want to optimise each user’s utility from each interaction with the system. Recent works have proposed new learning and evaluation methods for the ESR setting \[108, 45, 115, 49\]. One challenge that has been identified is handling of many objectives (more than three). This issue would be important to address in future research on reinforcement learning for simulation-based training, where there are multiple objectives related to the simulated scenario as well as the performance of the trainee.

A related line of future research is improving reward functions for each objective. For instance, in this dissertation we have studied several scenarios where one of the objectives is related to risk management. In our initial experiments we have used very simple models of risk, e.g. the distance between an aircraft and an air defence system. In future work it would be valuable to incorporate more advanced models of risk in the agent’s reward function. For instance, models that can estimate a fighter aircraft’s survival probability in each environment state. In combination with a non-linear utility function, such a risk model could lead to more realistic behaviour than when simply balancing reward signals with preference weights for, e.g., time and average safety.

Model-Based Multi-Objective Multi-Agent Learning

Our work on MO-Dreamer studied model-based multi-objective reinforcement learning in environments with dynamic utility functions, but only with one learning agent. To support simulation-based training, the method should be extended to multi-agent systems with multiple learning agents. In this setting agents would need a capacity to model other agents’ decision-making, in addition to learning about the transition dynamics and reward signals of the
8.3. Future Work

Future Work

environment alone. MO-Dreamer could probably be a good basis for this work, since the Dreamer world model has proven successful in many settings. The Dreamer agent has also been extended to single-objective cooperative multi-agent settings, with good results.

One possible approach would be to extend MO-Dreamer’s world-model to support self-play, which could enable learning of capabilities that are important in air combat simulation. For instance, the world model could be used to train agents in identifying other agents’ utility functions. Such a capability could support decision-making in cooperative as well as competitive settings, by allowing agents to make predictions about other agents’ future actions.

Environment Generation and Adaptation

The development of agents in part II of the dissertation focused on the Synthetic Trainer Agent. To improve the adaptive qualities of a training system, future work should investigate designs suitable for the Scenario Adaptation Agent. Here, it is possible that some components could be shared between the agents that participate as actors in training scenarios and the agents that work offline to adjust the training environment. For instance, the capability to construct models of other agents would be useful to support evaluation of trainees in different learning objectives. This modelling capability should include the capability of learning other agents’ utility functions, so that they do not have to be provided by an external entity, as we assumed in part II of the thesis. Another necessary extension is to study adaptation to a group of users with different utility functions, instead of a single user. That would require studying the group’s collective utility, and finding a balance between the individual utility functions of the participants. As a first step, methods for curriculum learning could be developed and evaluated in an environment with a population of synthetic pilots. These methods could then be adapted to pilot training systems and evaluated with humans in the loop, as discussed in the next section.

Experiments with Human Participants

In our experimentation with learning agents in this dissertation, only very limited experiments with humans in the loop were conducted. Further investigations of human-agent interaction would be highly valuable. To make it possible to study relevant training scenarios, a suitable approach could be to combine traditional methods for behaviour modelling with modern machine learning techniques. For instance, a handcrafted behaviour tree for a synthetic fighter pilot could be used as a starting point, and then successively updated to include an increasing number of nodes realised with reinforcement learning. As discussed in Chapter 5, this type of structured behaviour model could make each learning task easier compared to learning a complete,
8. Conclusion

complex behaviour from scratch. By starting from a behaviour model used in operational training, it becomes easier to evaluate new techniques for behaviour modelling in real training sessions, and more data can be collected to support the evaluations.

8.4 Final Words

In addition to the specific future research directions presented in Section 8.3, we recommend further user-centred development of learning agents, to steer progress in the most valuable direction. Real-world air combat scenarios provide many challenges to agents, e.g., cooperation and competition in scenarios with many human and synthetic agents, decision-making under partial observability and uncertainty, and the need to prioritise among multiple conflicting objectives, such as tactical mission goals, resource consumption, and safety. Therefore, the domain of air combat training is an excellent benchmark for reinforcement learning algorithms, and there are many exciting directions of research left to explore.
Bibliography


### Department of Computer and Information Science

Linköpings universitet

### Dissertations

**Linköping Studies in Science and Technology**

**Linköping Studies in Arts and Sciences**

**Linköping Studies in Statistics**

**Linköping Studies in Information Science**

<table>
<thead>
<tr>
<th>No</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>A Program Manipulation System Based on Partial Evaluation</td>
<td>Bengt Magnahagen</td>
</tr>
<tr>
<td>17</td>
<td>Probability Based Verification of Time Margins in Digital Designs</td>
<td>Bengt Magnahagen, Herbert C. T. Reif, and K. S. Narendra</td>
</tr>
<tr>
<td>18</td>
<td>Semantic analyses of process-behavioring in natural language systems</td>
<td>Mats Cedwall, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>22</td>
<td>A Machine Independent LISP Compiler and its Implications for Ideal Hardware</td>
<td>Per A. van Oorschot, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>33</td>
<td>Compilation of Multiple File Queries in a Meta-Database System</td>
<td>Tore Risch</td>
</tr>
<tr>
<td>51</td>
<td>Synthesizing Database Structures from a User Oriented Data Model</td>
<td>Mats Cedwall, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>54</td>
<td>Contributions to the Development of Methods and Tools for Interactive Design of Applications Software</td>
<td>Anders Haraldsson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>58</td>
<td>Performance Enhancement in a Well-Structured Pattern Matcher through Partial Evaluation</td>
<td>Bengt Magnahagen, Bengt Andersen, and Per A. van Oorschot</td>
</tr>
<tr>
<td>69</td>
<td>A Specification of an Abstract Prolog Machine and its Application to Partial Evaluation</td>
<td>H. Jan Komorowski, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>71</td>
<td>Knowledge Engineering Techniques and Tools for Expert Systems</td>
<td>René Reboh, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>77</td>
<td>Mechanisms of Modifiability in Large Scale Software Systems</td>
<td>Östen Oskarsson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>94</td>
<td>Code Generator Writing Systems</td>
<td>Hans Linnell, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>97</td>
<td>Advances in Minimum Weight Triangulation</td>
<td>Andrzej Lingas, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>109</td>
<td>Towards a Distributed Programming Environment based on Incremental Compilation</td>
<td>Peter Fritzson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>111</td>
<td>The Design of Expert Planning Systems</td>
<td>Erik Tengvold, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>155</td>
<td>Heuristics for Minimum Decompositions of Polygons</td>
<td>Christos Levcopoulos, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>165</td>
<td>A Theory and System for Non-Monotonic Reasoning</td>
<td>James W. Goodwin, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>170</td>
<td>A Formal Methodology for Automated Synthesis of VLSI Systems</td>
<td>Zebo Peng, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>174</td>
<td>A Paradigm and System for Design of Distributed Systems</td>
<td>Johan Fagerström, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>192</td>
<td>Towards a Many Valued Logic of Quantified Belief</td>
<td>Dimitar Driankov, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>213</td>
<td>Non-Monotonic Inheritance for an Object Oriented Knowledge Base</td>
<td>Lin Padgham, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>214</td>
<td>A Formal Hardware Description and Verification Method</td>
<td>Tony Larsson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>221</td>
<td>Fundamentals and Logical Foundations of Truth Maintenance</td>
<td>Michael Reinfrank, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>239</td>
<td>Knowledge-Based Design Support and Discourse Management in User Interface Management Systems</td>
<td>Jonas Löwgren, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>244</td>
<td>Meta-Tool Support for Knowledge Acquisition</td>
<td>Henrik Eriksson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>252</td>
<td>An Epistemic Approach to Interactive Design in Multiple Inheritance Hierarchies</td>
<td>Peter Eklund, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>258</td>
<td>A Non-Monotonic Formalism with Explicit Defaults</td>
<td>Patrick Doherty, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>260</td>
<td>Generalized Algorithmic Debugging</td>
<td>Nahid Shamsheer, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>264</td>
<td>Representation of Discourse-Cognitive and Computational Aspects</td>
<td>Nils Dahlbäck, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>265</td>
<td>Abstract Interpretations and Abstract Machines: Contributions to a Methodology for the Implementation of Logic Programs</td>
<td>Ulf Nilsson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>270</td>
<td>Theory and Practice of Tense-bound Object References</td>
<td>Ralph Rönquist, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>273</td>
<td>Pipeline Extraction for VLSI Data Path Synthesis</td>
<td>Björn Fjellborg, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>276</td>
<td>A Formal Basis for Horn Clause Logic with External Polymorphic Functions</td>
<td>Staffan Bonnier, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>277</td>
<td>Developing Knowledge Management Systems with an Active Expert Methodology</td>
<td>Kristian Sandahl, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>281</td>
<td>Compositional Modelling and Frame-Work for Verification</td>
<td>Christer Räckström, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>292</td>
<td>Studies in Incremental Natural Language Analysis</td>
<td>Mats Wärn, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>297</td>
<td>Interprocedural Dynamic Slicing with Applications to Debugging and Testing</td>
<td>Mariam Kamkar, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>302</td>
<td>A Study in Diagnosis Using Classification and Defaults</td>
<td>Tingting Zhang, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>312</td>
<td>Dialogue Management for Natural Language Interfaces - An Empirical Approach</td>
<td>Arne Jönsson, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
<tr>
<td>338</td>
<td>Reactive Systems in Physical Environments: Compositional Modelling and Frame-Work for Verification</td>
<td>Simin Nadji-Tehrani, Bengt Magnahagen, and Herbert C. T. Reif</td>
</tr>
</tbody>
</table>
Perspective, Development and Evaluation of the Training from an Organisational Learning
Jaime Villegas:

Studies in Action Planning:
Peter Jonsson:
Directional Types in Logic
3.

Hua Shu:

Johan Boye:

Jaime Villegas: Simulation Supported Industrial Training from an Organisational Learning


Kjell Orsborn:


Fredrik Nilsson:

Mikael Lindvall:

Mikael Ericsson:

Lars Karlsson:


Jörgen Hansson:

Niklas Hallberg:


<table>
<thead>
<tr>
<th>No</th>
<th>Namn</th>
<th>Titel</th>
<th>År</th>
<th>ISBN</th>
</tr>
</thead>
</table>