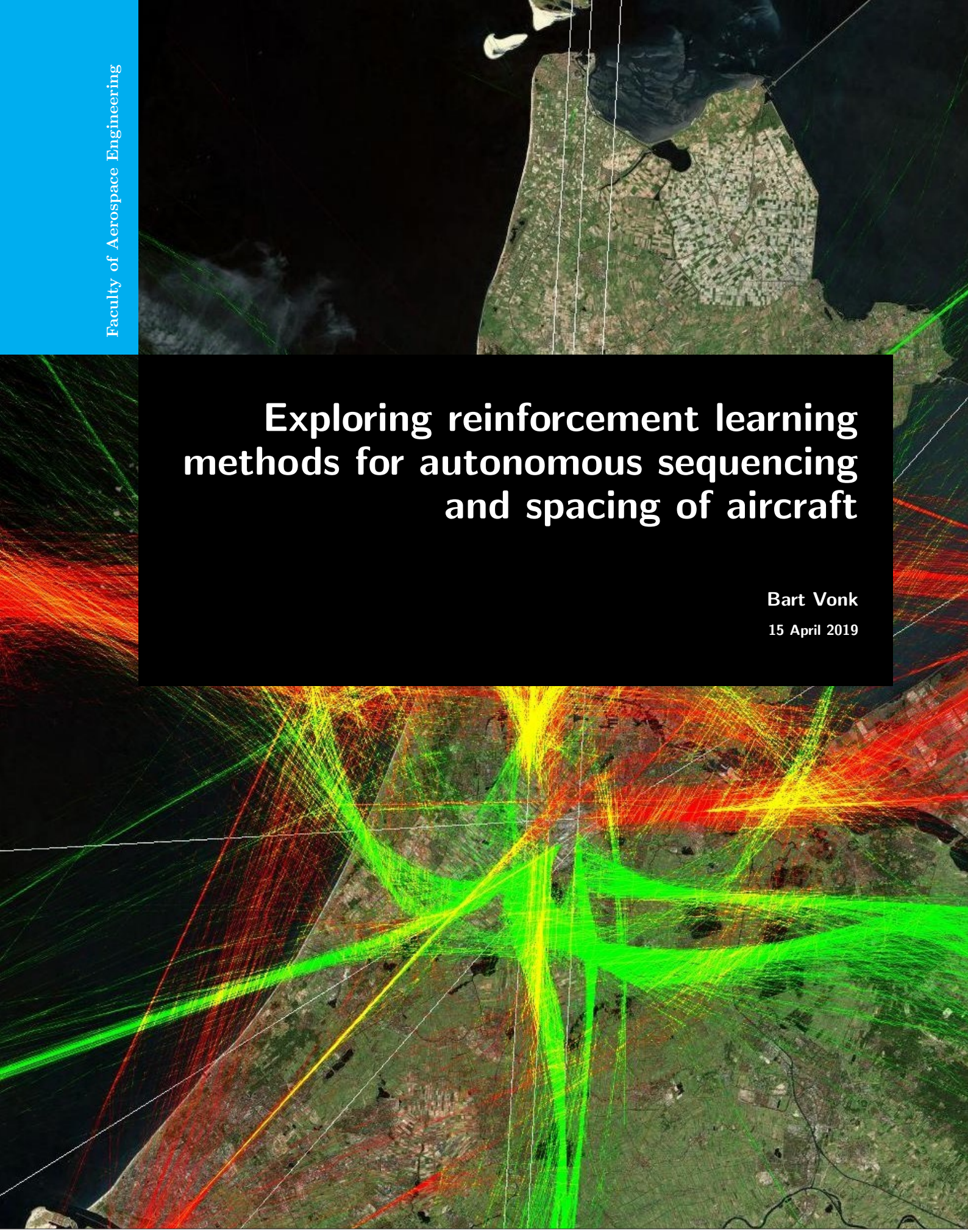
Faculty of Aerospace Engineering



Exploring reinforcement learning methods for autonomous sequencing and spacing of aircraft

Bart Vonk

15 April 2019

TU Delft Delft University of Technology

Challenge the future

Exploring reinforcement learning methods for autonomous sequencing and spacing of aircraft

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Bart Vonk

15 April 2019

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Exploring reinforcement learning methods for autonomous sequencing and spacing of aircraft"** by **Bart Vonk** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 15 April 2019

Readers:

Dr. ir. J.M. Hoekstra

Dr. ir. J. Ellerbroek

Dr. M.A. Mitici

Contents

I Scientific Paper **3**

II Preliminary Thesis **21**

III Appendix **101**

A Trajectories experiment 3 **103**

Preface

This work concludes my time at the TU Delft as a student. I would like to take the opportunity to say thank you to a few people. First of all I'd like to thank Jacco Hoekstra and Joost Ellerbroek for their feedback and guidance during my thesis. I would also like to thank my fellow students from room Sim 0.008 for their pleasant company and support.

I also want to thank Gina for helping me with my presentations and proof-reading. Finally, I would like to thank my parents for their support during my time as a student and giving me the freedom to explore my interests.

Bart Vonk

Report Outline

The outline of this report is as follows. Part I of the report is the scientific paper that has been written to show the results of the experiments performed to answer the research question: 'Which methods are suitable to automate aircraft sequencing and spacing tasks when a controller is trained using reinforcement learning?'. Part II of the report is the preliminary thesis, which includes the literature study as well as preliminary research and the research plan. Finally, additional results that are not in the paper can be found in the appendix.

Part I

Scientific Paper

Exploring reinforcement learning methods for autonomous sequencing and spacing of aircraft

B. Vonk
Supervisors: J. Ellerbroek, J.M. Hoekstra
Delft University of Technology, Kluyverweg 1, Netherlands

***Abstract*— Research on reinforcement learning algorithms to play complex video games have brought forth controllers surpassing human performance. This paper explores the possibilities of applying these techniques to the sequencing and spacing of aircraft. Two experiments are performed. First a single aircraft must learn to fly a 4D trajectory using only heading commands. To train an agent Duelling Deep Q-Networks has been applied to learn a successful policy, however, learning is unstable and does not provide a suitable basis for extending this to a multi-agent setting. Second a multi-agent experiment is performed where aircraft have to sequence and space themselves for landing without a 4D constraint. A Bidirectional Communication Net has been trained using Deep Deterministic Policy Gradients first on a single traffic scenario and then on multiple traffic scenarios. Emerging strategies have been seen in the single scenario training eg. a holding, but no optimal policy was found. Training on multiple traffic scenarios showed no coordination efforts between the aircraft. Further analysis showed the importance of a proper reward function and exploration strategies which were likely the cause of not finding an optimal policy for a multi-agent setting.**

I. INTRODUCTION

Terminal airspace is characterized by the need for many tactical decisions by air traffic controllers (ATCos). This is due to high density traffic situations where aircraft flows must be merged into landing sequences while also maintaining safe separation. Additionally, the terminal area is low altitude airspace, which can be subject to restriction such as aircraft noise regulations and adverse meteorological conditions. Separation is assured by vector commands from ATCos. As air travel has become a commodity due to cheap ticket fairs, the demand for air travel has increased [1]. Also, the number of minimum separation violations in Dutch airspace has grown. Most occurrences appear in the flight phase just before or during approach [2]. Hence, there is room for improvement in the sequencing and spacing of air traffic in terminal airspace.

A way of improving sequencing and spacing operations is to restructure the terminal airspace to better accomodate the process like with the Point-Merge (PM) system [1]. Other solutions to automate the sequencing and spacing use algorithms for conflict detection and resolution. [2], [3], [4]. Conflicts are probed based on 4D trajectories. Then resolutions are proposed based on a set of aircraft manoeuvres from which the best manoeuvre is chosen to resolve the conflict.

Engineering a system like PM is essentially designing the solving strategy for the sequencing and spacing problem of aircraft. Could it instead be possible to have an algorithm learn how to efficiently solve the sequencing and spacing problem autonomously to learn from the emerging strategies? Developments in Artificial Intelligence have shown that an agent can learn to play complex video games in an unsupervised manner without human demonstration with emerging performance surpassing human performance. For instance on a simple game like Breakout [5] to more recently very complex games that take humans years to master like Dota 2 [3]. The machine learning technique used in learning these games is reinforcement learning.

To surpass human-level performance on Breakout the Deep Q-Networks (DQN) algorithm was developed, which used neural networks as a function approximator for the value of the state-action space. Many improvements have been made on the DQN algorithm since it was introduced. Improvements were based on the double Q-learning algorithm [6], [7] or network architecture [8]; sampling and training techniques [9] and an emperical study of combinations of mentioned improvements [10]. Next to that other traditional reinforcement learning algorithms have also been extended with neural network function approximators such as Deep Deterministic Policy Gradients (DDPG)[11] and Asynchronous Actor Critic (A3C) [12].

An ATCo must, however, control multiple aircraft. The sequencing and spacing problem is thus a multi-agent problem. One of the assumptions underlying reinforcement learning is that the environment in which the agent is learning is stationary. When extending reinforcement learning algorithms to a multi-agent setting in a naive way the problem arises that training is unstable due to the fact that the environment becomes non-stationary from the perspective of a single agent. Solutions to this issue involve using centralized policies during training and decentralized execution at test time [13]. This means that during training the agents have access to each others policy so they know how other agents evolve. Other solutions use communication between agents to remove the non-stationarity issue [14], [15]. An advantage of a continuous communication channel is the possibility of having a variable number of agents present at runtime.

[1] Annual Report Schiphol Group 2017
[2] Annual Report LVNL 2016

[3] OpenAI research team, 'OpenAI Five', 2018, url: `https://blog.openai.com/openai-five/`, [accessed: June 25, 2018]

Related work in applying reinforcement learning in air traffic management used a reinforcement learning controller to control the 'time in trail' between arriving aircraft on the merge point of the PM system. When looking particularly at sequencing and spacing a step towards this objective is taken by [16] who used a hierarchical controller approach to solve the sequencing and spacing of two aircraft in the game Sector 33 by NASA. The hierarchical controller is build up on an outer and an inner controller. The outer controller chooses the aircraft route from a set of available routes at the start of the episode after which it cannot be changed. The inner controller then controls the aircraft speed during the episode. Limitations of this approach are the fixed approach set of aircraft and aircraft trajectories that are embedded in the action design. Hence, the solution cannot be used for any new situations.

This paper aims to explore the possibilities to apply reinforcement learning techniques to the sequencing and spacing of aircraft in the TMA and work towards discovery of more efficient and safe operation strategies in the TMA. Therefore the question: 'Which methods are suitable to automate sequencing and spacing tasks when a controller is trained using reinforcement learning?', must be answered. To this end multiple reinforcement learning techniques have been implemented and tested on different environments and traffic scenarios.

II. BACKGROUND

A. Reinforcement Learning

Reinforcement Learning is a class of machine learning used to solve sequential decision making problems and is modelled as a Markov Decision Process (MDP). In short in a MDP the goal of an agent is to maximize the reward it receives from an unknown environment with which it interacts by performing actions. A MDP is represented by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ where:

- $\mathcal{S}$ is the set of states.
- $\mathcal{A}$ is the set of actions.
- $\mathcal{P}$ is the transition probability that action a_t in state s_t will lead to state s_{t+1} at the next timestep.
- r is the immediate reward received by the agent for the state transition from s_t to s_{t+1} by performing action a_t.
- γ is the discount factor, which represents the relative importance of future and immediate rewards.

Using this framework the actions an agent takes in the environment is represented by the policy π, which is thus a mapping from state to action. The optimal policy is represented as the maximum of the sum of expected rewards over the course of an episode in the environment. The process is visually shown at Figure 1. The agent observes state s_t and reward r_t and at timestep t, chooses an action a_t according to its policy. The action changes the state of the environment in the next timestep $t+1$. This process repeats itself during an episode until the episode terminates.

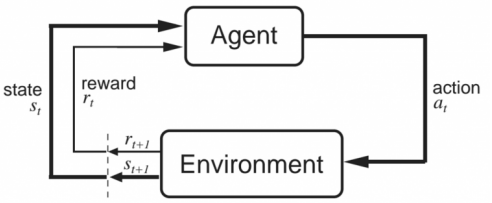


Fig. 1: Block diagram describing the reinforcement learning process. An agent observes the current state and reward in an environment. According to its policy the agent selects an action, which changes the state of the environment and the agent observes the reward and state in the next timestep.

B. Q-learning

Q-learning, introduced by Watkins [17], is one of the most fundamental reinforcement learning algorithms, and is classified as a value-based method. This means that it tries to learn how valuable it is to take a particular action in a particular state by learning the Q-value of every state-action combination in the environment. This Q-value represents the expected value of the discounted cumulative reward for that action when following policy π in Equation 1 with k the number of steps in the future and the current timestep t, which must be learned for each state-action pair.

$$Q(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s, a \right] \tag{1}$$

The value of each state-action pair can be stored in a Q-table for discrete state-action spaces. For continuous state spaces function approximation with a Q-function is used. When taking a greedy approach the policy is $\pi(s) = \arg\max Q(s,a)$, which takes greedily takes the action yielding the highest expected cumulative reward. The agent can learn about the value of each state-action pair by visiting all of them a sufficient number of times and observing the reward from the environment, after which the following update step is applied in Equation 2 with learning rate α and where state and action in the next timestep will from now on by indicated by s' and a' respectively. To ensure all states are visited the agent explores the environment by sometimes choosing a random action with probability ϵ. This is called an ϵ-greedy policy. The process of exploring the environment is called exploration.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max Q(s',a') - Q(s,a)) \tag{2}$$

Q-learning is only suitable for discrete action spaces, due to the fact that it compares the Q-value for multiple actions to choose an action.

C. Deterministic Policy Gradient

Another family of RL algorithms are formed by the policy gradient algorithms, which do not learn value but directly try to find an optimal policy. Where in Q-learning the policy is to ϵ-greedy choose the highest Q-value in a state in policy

gradient the policy has parameters and is thus a function. A policy is parametrized with parameters θ and the action output is modelled as a probability distribution over the actions $\pi(a|s,\theta)$. An objective function $J(\theta)$ is specified which is the equivalent of the cost function in an optimization process, see Equation 3 where $\rho_{a_\theta^T}$ is the discounted state distribution [18]. The discounted state distribution describes the probability that an agent will visit state s after when executing policy π. The objective function is used to optimize the policy for the objective using gradient ascend. The gradients are computed by taking actions according to the policy and observing the reward. The policy π is stochastic because the action output is a probability distribution over the actions. This is also necessary to properly explore the environment.

$$\begin{aligned} J(\theta) &= \int_S \rho_{\pi_\theta}(s) r(s, \mu_\theta(s))\, ds \\ &= \mathbb{E}_{s\sim\rho_{\pi_\theta}}[r(s, \mu_\theta(s))] \end{aligned} \tag{3}$$

Note that a deterministic policy gradient (DPG) also exists [19]. Instead of using a stochastic policy $\pi\,(a|s,\theta)$ actions are selected deterministically using policy $\mu(s,\theta)$. The DPG is a limiting case of the stochastic policy gradient when the variance goes to zero. However, to properly explore the environment exploration noise must be added, making the policy stochastic again. Therefore the DPG is often implemented as an actor-critic method to allow for off-policy exploration. Actor-critic architecture will be discussed next. Off-policy learning is when policy updates are not computed using actions sampled from the current policy. Hence, noise can be added to the action output for additional exploration without creating a stochastic policy. An advantage of policy parametrization over action-value modelling is that the form of the policy can be used to include knowledge into the learning system [20].

D. Actor-Critic

Actor-critic methods are a combination of a policy gradient method and value based method and are separated into an actor and a critic [18]. The critic learns a value function that takes as input the action taken and the state and produces a Q-value which is used to 'criticize' the value of the state-action pair. The actor is a parametrized policy as in the policy gradient. Actor-critic is thus a combination of a value based method and policy gradients. The critic estimates the error that is used to update the parameters of the actor, see Figure 2. The actor does not have direct access to the reward signal.

By using this construction of decoupling the action selection from learning a value function the variance of the policy gradients is reduced at the cost of introduction of a bias due to the approximation of the policy gradient with the use of a value function. Policy gradients are known to have high variance due to the way they are computed. The gradients are computed by a randomly sampling a trajectories during an episode. The computation of the gradient relies on these

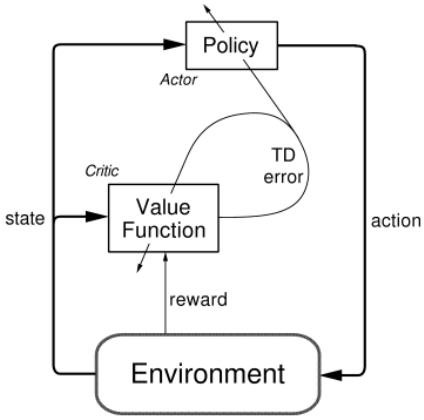


Fig. 2: Actor-critic architecture. The critic learns a value function which it uses to 'criticize' the action taken by the actor. The critic delivers a gradient the actor which is used to update the actor's policy. The actor has no direct access to the reward received from the environment.

random sampled trajectories and therefore have variance. In other words, the computed gradient does not always point in the right direction. In actor-critic methods the actor updates the policy in the direction suggested by the critic which learns a value function similar to Q-learning, which has lower variance. The reduction in variance of the policy gradient means fewer samples are required for learning.

E. Deep reinforcement learning

Deep reinforcement learning uses neural networks from the field of deep learning as a non-linear function approximator for the parametrized policy and the value function. Deep neural networks are useful because they are very versatile in their architecture and applications. They are used in the field of supervised learning for applications such as function approximation, image classification, and natural language processing. These allow each different sensory inputs, such as values, pixel data and sequence data respectively.

In DQN a neural network is used to approximate the Q-function [5]. The application of neural networks as function-approximator allows for end-to-end learning. This means that it directly maps sensory inputs to action outputs, e.g. image frames to actions. However, when using a non-linear function approximator to learn state-action values reinforcement learning is unstable [21]. The instability is caused by correlation in sequential observations and correlation between action values and the target value for Q-learning. To stabilize the training of neural networks in a reinforcement learning setting additional measures are introduced. A replay memory is introduced, which is filled with experiences. At every timestep the experience (s_t, a_t, r_t, s_{t+1}) is saved in the replay memory. Experiences are sampled randomly from the replay memory to prevent temporal correlation in training data. To decorrelate the Q-values and the target Q-values a separate target network is introduced which estimates the target value. The target network is only updated every c

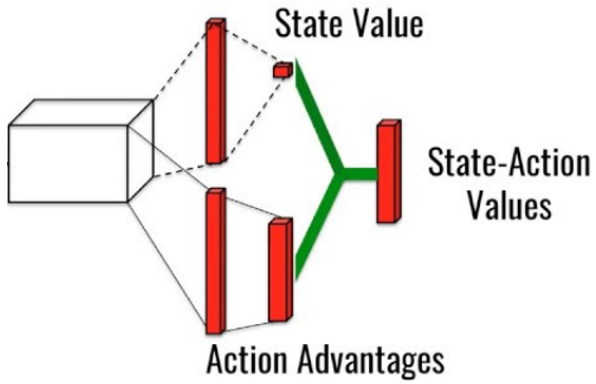


Fig. 3: Dueling-DQN network architecture. The network is separated into a value stream and an advantage stream, which combined are equal to the state-action values (Q-values). Advantage estimation implicitly learned in this network architecture. [8]

number of iterations to keep the optimization target for the network constant over a number of consecutive optimization epochs thereby reducing correlation with the Q-values.

A disadvantage of DQN is that difference between Q-values relative to their magnitude is often very small, which may result in poor action choices. Advantage estimation is used to subtract a baseline value from all Q-values in a given state without introducing a bias to obtain a larger differences in Q-values relative to their absolute value. Advantage is simply the Q-values subtracted by a common baseline $A(s,a;\theta) = Q(s,a;\theta) - V(s;\theta)$. This advantage is then used as Q-value estimate rather than the actual Q-value in the Q-learning update step. The relative difference between the advantage estimates is larger than the relative difference between Q-values and thereby speeds up learning. [22]

Dueling-DQN [8] incorporates this principle in its network architecture by splitting the network into two streams. The first estimates the state value $V(s;\theta)$ and the second the action advantage $(A(s,a;\theta)$ and merge them later to obtain the Q-values. Hence, Equation 4 is implicitly modelled in the network architecture to leverage advantage estimation and improve learning performance. The different streams in the network architecture are visualized in Figure 3.

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta \right) \tag{4}$$

F. Multi-agent reinforcement learning

Sequencing and spacing requires multiple aircraft to coordinate their actions. Reinforcement learning must be applied to a multi-agent setting, but doing so naively does not work. One of the assumptions underlying reinforcement learning is that the environment does not change, it is stationary. When multiple agents learn a policy independently this assumption is violated. From the perspective of a single agent the other agents are part of the environment, it does not know it shares the environment with other learning agents. When all agents update their policy independently based on their own observations and the rewards they have received the environment becomes non-stationary from the perspective of a single agent. This is because the behaviour of the other agents, which are part of the environment, changes with policy updates and subsequently the environment dynamics have changed.

The Bidirectional Communication Net (BiCNet) [15] solves this issue by representing all agents in a single policy network with shared parameters. The architecture of the neural network allows information to be exchanged between agents. BiCNet makes use of a recurrent neural network (RNN). A RNN is used to learn to predict sequence data (eg. a sentence or time series data). A sequence with length T_x is defined by a set of input vectors $\mathbf{x} = \{x^{<1>}, x^{<2>}, ..., x^{<T_x>}\}$ where $<>$ represents the sequence entry number. The RNN will output a sequence with length T_y. An RNN [23] takes two inputs, namely the entry of a sequence $x^{<t>}$ and hidden state $h^{<t-1>}$ at time step t where $h^{<0>}$ is initialized with a zero vector. It outputs a new hidden state $h^{<t>}$ and estimate $\hat{y}^{<t>}$. The hidden state contains the information passed from one sequence entry to the next Contrary to a feed-forward neural network, a RNN has multiple input and output vectors. Thus, every input/output pair has its own set of network weights. A RNN only allows information to be passed from the start of the sequence towards the end of the sequence. Hence, BiCNet uses a Bidirectional-RNN which processes the sequence in both directions and adds the output estimates so information is exchanged in both directions.

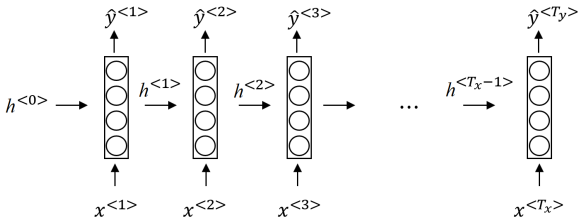


Fig. 4: Unrolled recurrent neural network that shows the information flow trough the network of a sequence x with length Tx. All neurons shown in the image use the same set of network weights.
[4]

BiCNet is built up of three parts. First there is a preprocessing layer of the network which is the same for all agents. The preprocessed observations are then passed through an N-to-N bidirectional recurrent neural network which outputs the entire sequence again. N-to-N means that the input sequence length $T_x = T_y$. This layers acts as a communication channel between agents so the agents can learn an effective communication strategy. Thirdly the sequence is post-processed similar to the pre-processing

[4] Andrew Ng, 'Coursera Deep Learning Specialization - Sequence Models', [Lecture Notes]

layers to output an action for each agent. A key advantage of this architecture is that the number of agents can vary at runtime, because the network maps any number of agents to the same number of aircraft.

To train BiCNet an actor-critic architecture is used along with the DPG, where both the actor and the critic have the BiCNet network architecture. The objective function for a multi-agent setting where all agents are collectively represented in a single policy becomes a sum over the rewards of N agents, see Equation 5. Each agent receives individual rewards.

$$J(\theta) = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\sum_{i=1}^{N} r(s, a_\theta(s))\right] \tag{5}$$

Then according to the Multiagent Deterministic Policy Gradient Theorem where agents are collectively represented in a policy, the policy gradients is as in Equation 6 [15], with N the number of agents.

$$\nabla_\theta J\theta = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\sum_{i=1}^{N}\sum_{j=1}^{N} \nabla_\theta a_{j,\theta}(s) \cdot \nabla_{a_j} Q_i^{a_\theta}(s, a_\theta(s))\right] \tag{6}$$

III. Experiment 1: Single-agent simulation

Existing sequencing and spacing automation systems are built up of a number of components. A 4D trajectory predictor to predict the time of arrival for aircraft at key points along the route. A second component is a controller that chooses manoeuvres (e.g. a path stretch manoeuvre) for an aircraft to perform to arrive at the correct time as assigned by the third component, the Arrival Manager(AMAN), or to keep spacing with other aircraft.

To find emerging strategies for sequencing and spacing this setup is used for which an aircraft is assigned a scheduled time of arrival (STA). The goal of the experiment is then to simultaneously learn to navigate to the final approach fix (FAF) while also arriving at the correct time as to mimic the interaction with the arrival manager.

A. Environment description

The experiment is performed using the BlueSky Open Air Traffic Simulator [24]. At the start of an episode an aircraft is initialized at initial approach fix (IAF) SUGOL with heading to equal to the bearing towards the FAF at waypoint EH609, see Figure 5. At the start of the episode a delay time is sampled from a uniform distribution in the range $[0, 100]$ seconds that the agent has to absorb to arrive at FAF on the STA. The episode terminates if the FAF is reached or if the absorbed delay falls below a negative threshold. The aircraft has three states available for observation. The distance d to the target waypoint, the relative ψ_{rel} to the track angle to the target waypoint and the delay t_{delay} that must still be absorbed along the route This state is computed using a basic 4D trajectory predictor which calculates the fastest possible flight path to the target waypoint with the current speed

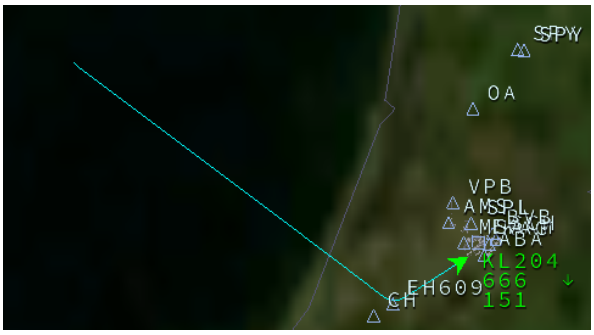


Fig. 5: The environment for the single agent experiment. An aircraft is initialized at SUGOL and must fly to EH609 to make an approach for Schiphol RWY06

and uses the information to compute the excess time which still must be absorbed along the route. The action space is discrete, the agent can turn left, right or do nothing. The aircraft will then make the maximum turn possible within one timestep.

$$\mathcal{S} = [d, t_{delay}, \psi_{rel}]$$
$$\mathcal{A} = \{\text{left}, \text{straight}, \text{right}\}$$

The reward function in Equation 7 is built up of three components: distance, time, and a binary reward based on a time derivative. If the episode terminates without reaching the goal, a penalty is given via the t_{rew}. The distance is added as a shaped reward to give the agent an indication of where to go. The time reward is added to promote the absorption of time. Finally the binary reward based on the change of t_{delay} in between states is used to encourage good behaviour once the agent overshoots the t_{delay}. If the episode terminates without reaching the FAF a penalty is given.

$$R = (3 + \alpha_d d) + t_{rew} + dt_{rew}\Delta\psi$$

$$\text{where } t_{rew} = \begin{cases} -100 & \text{if term. penalty} \\ 10 + \alpha_t & \text{if not term. penalty} \end{cases},$$

and

$$\text{where } dt_{rew} = \begin{cases} -1 & \text{if } (t > 0 \text{ and } dt > 0) \text{ or} \\ & \text{if } (t \leq 0 \text{ and } dt > 0) \\ 1 & \text{if } (t > 0 \text{ and } dt \leq 0) \text{ or} \\ & \text{if } (t \leq 0 \text{ and } dt \leq 0) \end{cases} \tag{7}$$

TABLE I: Reward function parameters

α_d	-0.22
α_t	-0.2
α_{hdg}	-0.07

B. Model and training

To model the discrete action space in a continuous state space the Dueling-DQN network architecture is chosen. Subsequently the model will be trained using the DQN

algorithm listed in the Appendix as algorithm 1, which makes use of experience replay and a separate target network. The network summary is found in Table II. The network is a fully connected neural network. The Rectified Linear Unit is chosen as activation function, because of its computational efficiency. The output layer of the Advantage estimation stream 'Advantage 3' has 3 neurons corresponding to the size of the action space of the environment. Analogously the output of the Value stream 'Value 3' has 1 neuron corresponding to the state value. Both these layers have a linear activation function because they are estimating values that are not restricted to a particular range. In the output layer of the network the advantage prediction and value prediction are added to output a Q-value prediction. Finally to train the network a mean squared error loss function is used and to perform gradient descent the Adam optimization scheme is used. During the optimization the computed gradient is clipped between a minimum and maximum value as it was empirically found that this adds stability to the training process [5]

TABLE II: Dueling DQN network summary

Layer	Activation	Neurons	Connected to
Advantage 1	ReLu	128	Input
Advantage 2	ReLu	128	Advantage 1
Advantage 3	Linear	3	Advantage 2
Subtract mean	-	-	Advantage 3
Value 1	ReLu	128	Input
Value 2	Relu	128	Value 1
Value 3	Linear	1	Value 2
Add outputs	-	-	Subtract mean, Value 3
Loss function	Mean squared error		
Optimizer	Adam		

During training it is important the agent properly explores the state space to visit a large diversity of state-action combinations. Once the agent has acquired sufficient knowledge about the environment it must start exploiting its acquired knowledge to strengthen it in the network. To explore the environment an ϵ-greedy action selection strategy is taken. The agent will greedily choose the action with the highest q-value, but with probability ϵ a random action is chosen. Over time ϵ is annealed to move from exploration to exploitation of the learned policy. An overview of all hyperparameters used in training is given in Table III

C. Results

Numerous training runs with different hyperparameters have been attempted. The most successful run is shown here. Hyperparameters used are listed in Table III.

TABLE III: Hyperparameter overview for Experiment 1.

Replay memory size	2000
γ	0.98
ϵ_{start}	1.0
ϵ_{min}	0.01
ϵ_{decay}	0.99
λ	0.001
Batch size	32
c	1000
Gradient clip value	0.5

The training curve is shown in Figure 10 with a running average over 10 episodes to smoothen the curve. The agent quickly learns how it can obtain a high reward, after which performance oscillates and slowly with a downwards trend. The best performance is observed after approximately 250 episodes episode for which a 25 test runs with different initial delay times is performed. This does not coincide with the peak in the training curve in Figure 10. This can be explained by a combination of random exploratory action and a difficult initialization of the episode.

A boxplot of t_{delay} at termination in Figure 9 shows that for most initializations the learned policy is capable of arriving at the FAF at the STA of arrival with an error smaller than 5 seconds. The to be absorbed delay is plotted over time during these runs in Figure 8. This shows that the runs that have failed are those with an assigned delay smaller than 10 seconds. When inspecting the trajectories flow in Figure 7 it shows that successful trajectories offset from the nominal path and then fly towards the FAF in a curved path.

D. Discussion

As mentioned the agent learns a policy with which it performs well and achieves the intended goal of the experiment, but when the exploration of the agent stops and it starts exploiting the policy for learning (ϵ has been annealed to its minimum value), the agent unlearns its good behaviour, also called catastrophic forgetting. This training instability has been observed on all of the training runs performed in this experiment. This training instability was also observed when training on a simple environment where the agent must learn to balance an inverted pendulum. Therefore this phenomenon is not an artefact of the experiment environment. Stopping training early is thus necessary to obtain a satisfactory result, or subsequently save the network weights of intermediate solutions.

To add, many training attempts failed completely. Due to the reward of t_{delay} becoming negative once the agent overshoots the STA it becomes difficult for an agent to even learn where EH609 is located. Overshooting the STA is easily done by flying in the wrong directon. Terminating the episode as soon as the delay time has been surpassed resulted into learning 'suicidal behaviour'. The agent never discovered how it could obtain good rewards and instead learned to terminate the episode as quickly as possible by flying in the opposite direction. This yields high rewards in the short term because the state t_{delay} goes to 0 quickly, but once the aircraft overshoots the expected STA there is no

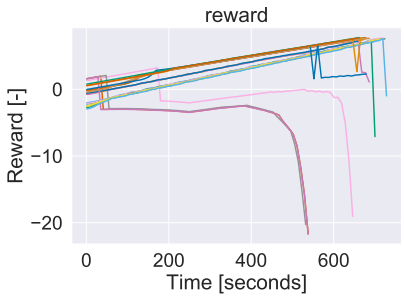


Fig. 6: Reward vs. time for 25 test episodes with network weights loaded after 250 episodes of training with Dueling-DQN

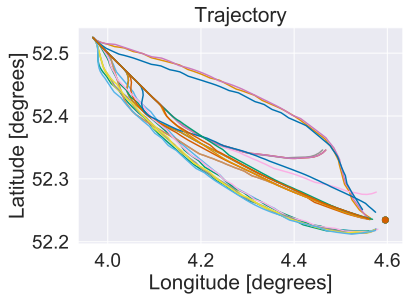


Fig. 7: Trajectories for 25 test episodes with network weights loaded after 250 episodes of training with Dueling-DQN

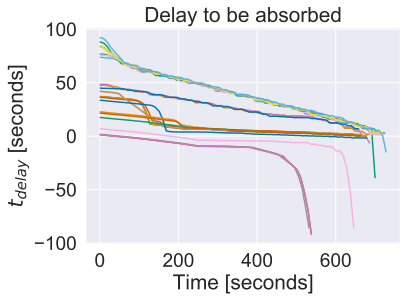


Fig. 8: Time to be absorbed for 25 test episodes with network weights loaded after 250 episodes of training with Dueling-DQN

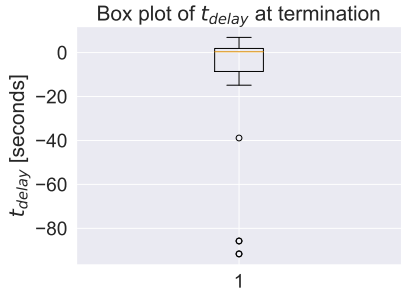


Fig. 9: t_{delay} at termination for 25 test episodes after 250 episodes of training with Dueling-DQN

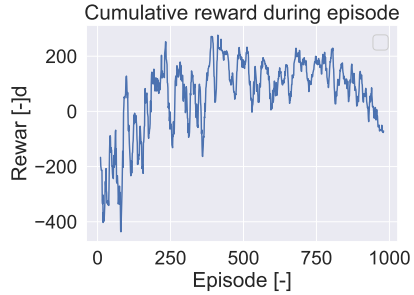


Fig. 10: t_{delay} at termination for 25 test episodes after 250 episodes of training with Dueling-DQN

way to recover because the aircraft is flying in the wrong direction. Thus, in the long run this is a bad policy. This shows a good exploration is key to having a successful training run as well as setting up a good reward function and termination conditions.

Next the strategies that have emerged to absorb the delay are discussed. The flown trajectories do not resemble manoeuvres a ATCo would issue. Instead the trajectories are more curve-like and most closely resemble a path-stretch manoeuvre. High penalties are given when an aircraft overshoots the STA. This causes the aircraft to fly a more conservative path rather than making aggressive manoeuvres right at the start of the episode. The trajectories also show oscillations in its path where the agent has sequentially chosen the actions left, right, left, right. This is an artefact of the discrete action representation. The policy failed for starting $t_{delay} < 10$ seconds, because this leaves very little room for error. Considering the oscillations present in the aircraft trajectory cost precious time in this case this could be a reason why it is difficult to learn a policy for this limiting case.

IV. Experiment 2: Multi-agent simulation

To model a complete sequencing and spacing system it is necessary to use a multi-agent setting. Training in a multi-agent setting adds complexity to the system as the state-action space grows rapidly. Training instabilities were already observed in the single agent experiment. These were partly accredited to the difficulties with the time reward in the 4D-trajectory optimization and artefacts of a discrete action space. Therefore, to remove complexity from the multi-agent setting these are removed.

For this part 3 separate experiments are performed using the same environment definition to find a suitable training method. The first to determine the reward function. The second to find emerging strategies with a single scenario and the third attempts to find a policy to generalize over multiple scenarios.

A. Environment Description

To simulation environment is again designed in the BlueSky Open Source Air Traffic Simulator [24]. An episode starts by loading a scenario file, initializing the aircraft. An episode terminates when all aircraft have landed, a loss of separation has occurred or the time limit of 175 timesteps with for the episode has been reached. Each timestep has $\Delta t = 15$ seconds. A time limit for an episode has been set to prevent infinite episodes when aircraft never land or collide. All aircraft fly at a constant speed of $200kts$ and constant altitude at FL200. Aircraft do not actually land due to the constant altitude flight, but are considered to have landed when they fly over AMS. The following states and actions

are available to the agent.

$$\mathcal{S} = \left[\text{lat}, \text{lon}, \text{hdg}_{\text{qdr}}, d, (\text{lat}_{\text{i}}, \text{lon}_{\text{i}}, \text{qdr}_i, \Delta d_i)_{i=1}^{n}\right]$$
$$\mathcal{A} = [\Delta\text{hdg}]$$

The states are latitude, longitude, the difference between the aircraft heading relative to the aircraft bearing to AMS in range $[-180, 180]$; and the distance to AMS. Additionally the agent observes a sequence of other aircraft including itself in case it is the only aircraft currently in the simulation. The action space is a heading change, which is a continuous variable in the range $[-\Delta\text{hdg}_{max}, \Delta\text{hdg}_{max}]$. 'Max' indicates the maximum possible heading change within one timestep to enable aircraft to make smooth turns over consecutive timesteps. This is dependent on the turning radius of the aircraft and aircraft speed, which are constant. To complete the environment description the reward function is required, which will be specified later.

B. Model and Training

One of the issues in a multi-aircraft simulation is that the number of aircraft at runtime is not fixed due to aircraft landing. This is an issue because the input size of a neural network is fixed and cannot be changed and hence the aircraft cannot be represented in a policy. However, by modelling the aircraft as sequence data, where each aircraft is a sequence entry, this issue can be bypassed as demonstrated with BiCNet. Therefore the policy network is designed along this principle.

The network architecture is shown in Figure 11. To ensure full state observability each aircraft agent has as input its own set of states and the sequence of other aircraft position relative to itself. Inputs are normalized to the range $[-1, 1]$ for faster training convergence. The aircraft relative position sequence passes through a fully connected hidden layer to preprocess the merged states after which a max-pool operation along the aircraft axis is performed for dimensionality reduction. A max-pool operation is a method for dimensionality reduction where the maximum value is taken along an axis to reduce the number of dimensions of a tensor. This is concatenated with the aircraft own state after which it is again passed through a fully connected hidden layer. The parameters of these pre-processing layers are shared among all aircraft. This is done for all aircraft after which the sequence of pre-processed data is passed through a bidirectional recurrent neural network, which acts as a communication layer between aircraft agents. Finally the processed sequence is passed through the output layer which also shares its parameters among the agents. Both the actor and the critic use the network. All hidden layers use a ReLU activation function except for the output layer. The output layer of the actor is a hyperbolic tangent. Using this activation function limits the output value range to [-1, 1] which is convenient for scaling the action output to $[-\Delta\text{hdg}_{max}, \Delta\text{hdg}_{max}]$. The activation function for the output layer of the critic is linear to output a real number for the Q-values.

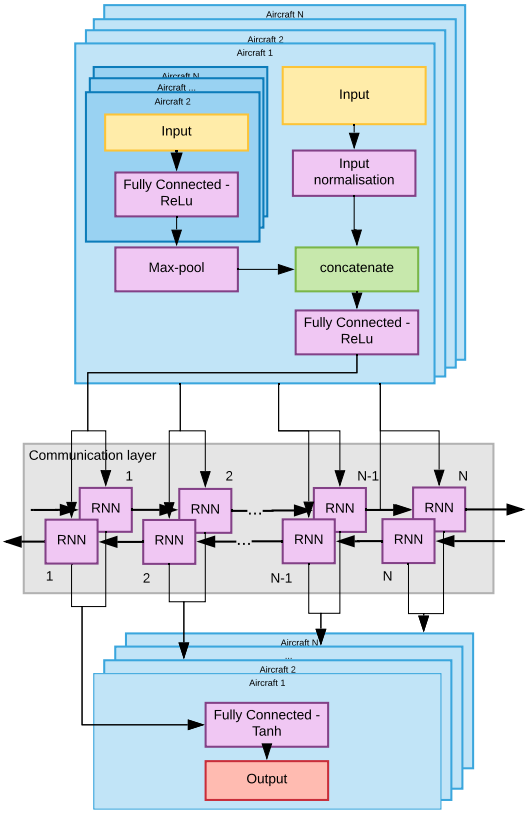


Fig. 11: Neural Network architecture for the experiment, the blue squares represent the aircraft sequence as input to the system. The yellow blocks are the aircraft states, the purple blocks are network layers. The green blocks are extra operations and the red block represents the network output.

For training of the model the DDPG extended with the Multi-agent Deterministic Policy Gradient Theorem as described earlier is implemented. The algorithm can be found in the Appendix at algorithm 2 The action output is deterministic so to explore the environment noise is added to the action outputs. Exploration noise is sampled from an Ornstein-Uhlenbeck (OU) process which satisfies Equation 8. An OU process is a random walk with the tendency to move back to a mean location. The long term mean μ random process dW_t and tuning parameters β and σ. OU noise is selected over Gaussian noise because the OU noise samples are correlated between timesteps. The dynamics of the aircraft position approach integrator dynamics with respect to the action and therefore act as a low-pass filter. Gaussian noise applied to action selection is thus filtered out by the aircraft dynamics over sequential actions whereas this is not the case for OU noise.

$$dX_t = -\beta(X_t - \mu)\, dt + \sigma\, dW_t \tag{8}$$

The training of neural networks occurs in batches. This is done to speed-up training and larger batch sizes provide a more accurate estimate of the gradient due to the reduction of variance. A batch of experiences is sampled from the replay

memory. However, not each sample in the batch contains the same number of aircraft. In order to use batch-processing efficiently all sequence samples must have equal length. To achieve this sequences within a batch are padded with zeros to equal length. During network inference and gradient computation these padded zeros are masked to prevent them from affecting the computed gradient and action values, while enabling their efficient computation.

TABLE IV: Actor network summary

Layer	Activation	Neurons	Connected to
Shared Dense	ReLU	32	Shared Input
Max pool	-	-	Shared TimeDistributed
Concatenate	-	-	Max pool, Local Input
Pre Bi-RNN	ReLU	32	Concatenate
Bi-RNN	ReLu	32	Pre Bi-RNN
Post Bi-RNN	Tanh	1	Bi-RNN

TABLE V: Critic network summary

Layer	Activation	Neurons	Connected to
Shared Dense	ReLU	32	Shared Input
Max pool	-	-	Shared TimeDistributed
Concatenate	-	-	Max pool, Local Input
Pre Bi-RNN	ReLU	32	Concatenate
Bi-RNN	ReLu	32	Pre Bi-RNN
Post Bi-RNN	Linear	1	Bi-RNN

TABLE VI: Hyperparameters used in the multi-agent experiments governing the reinforcement learning process.

Parameter	Value
BlueSky update interval	15
Memory length	10000
Batch size	32
γ	0.99
λ_{critic}	0.0005
λ_{actor}	0.0005
σ (OU)	0.2
θ (OU)	0.2
μ (OU)	0
dt (OU)	0.12
τ	0.9

C. Experiment A

1) Description: This experiment focusses on the design of the reward function. The reward function must reflect the learning goal for the agents. When the goal is to land aircraft at Schiphol a reward can be given each time an aircraft lands on Schiphol. This is a sparse reward as the agent only receives a reward when the aircraft lands and not in intermediate timesteps. The reward reflects the goal of the experiment, however, it may be so that the agent will fail to learn how to fly to Schiphol, because it never observes this reward during exploration when it cannot find the airport. Sparse rewards can therefore be a weak learning signal as it only receives reward once it has reached its goal. Many actions were taken to reach the goal so it is hard to assign credit to the actions that actually helped reaching the goal. A shaped reward on the other hand, is a reward structure that provides the agent with intermediate feedback guide it into the right direction. A shaped reward can speed up and stabilize learning. However, a shaped reward can also change the optimal policy because the agent now also starts to optimize to obtain these intermediate rewards.

This first experiments is conducted to determine whether a shaped reward function is necessary. Therefore two reward functions are designed of which one has an additional shaping component as shown in Table VII. A reward of 10 is given to an aircraft that has landed, a penalty of -0.05 at each timestep so the aircraft learns to fly the fastest route. A penalty of -5 is given to aircraft that violate the separation criteria. Finally the shaped component is a reward based on the heading of the aircraft to encourage it to align its heading with the bearing to AMS.

To leave multi-agent complexities out of this this experiment is performed with a single aircraft. At the start of each training episode an aircraft is initialized 120nm from AMS at a random direction with heading equal to the bearing to AMS. The aircraft must learn to navigate to AMS and fly the fastest route possible.

TABLE VII: Reward function components

Trigger	Value	Sparse	Shaped
Landed at AMS	10	x	x
Timestep	-0.05	x	x
Loss of separation	-5	x	x
Timestep	$0.2 - 0.4\left\lvert\frac{\text{hdg}_{\text{qdr}}}{180}\right\rvert$		x

2) Results: Three runs with different random seeds have been performed for both reward functions. The progress of episode length during training is shown Figure 12. The runs with shaped reward converge much faster to the optimal solution than the runs with sparse rewards. In one of the runs with sparse reward the agent never discovered AMS during exploration and hence failed to learn any useful behaviour. With the addition of loss of separation penalties in a multi-agent setting, it will be even more difficult to learn how to navigate to AMS without a shaped reward. Therefore the shaped reward is implemented in further experiments.

To gain further insight in what actions a single agent has learned to take across the state space, heatmaps are drawn to visualize the learned policy in Figure 13. As expected the agent has learned to choose actions to fly straight to AMS to take actions that will align its heading with the track to AMS.

D. Experiment B

1) Description: The goal for this experiment is for aircraft sequence and space themselves in a multi-agent setting when making an approach for AMS. Each episode 4 Aircraft are initialized in a symmetric diamond shape west of AMS. Aircraft must organize themselves into a landing sequence while maintaining safe separation. After every 25 episodes of training a test run is performed. Test runs are performed without exploration noise to test the performance of the deterministic policy.

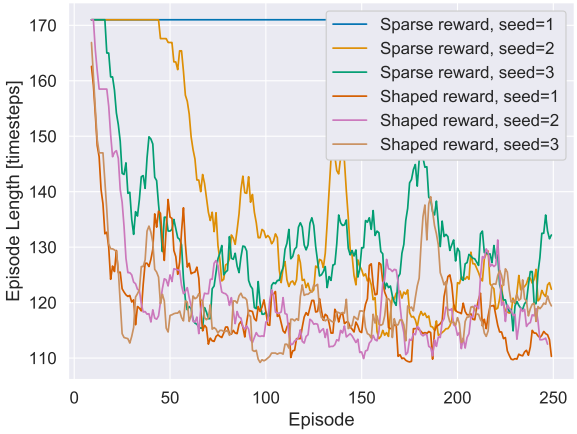


Fig. 12: Training progress for two reward functions with different random seed.

2) Results: The learning curve in Figure 14 shows that over time many feasible solutions have been found where no loss of separation occurs. If loss of separation has occurred the episode length is set to 200 in order to see the difference between successful episode a LOS or an episode that has reached its maximum time length. The learning curve also shows that the policy also moves away from these solutions again to finally result into catastrophic forgetting after approximately 40000 episodes after which all aircraft got stuck in a permanent right turn and never recovered.

Despite the algorithm not converging to the optimal solution, some interesting trajectories can be sampled from the successful runs in between, as shown in Figure 15. Aircraft will be referred to by their starting position being left, right, top, or bottom. In Figure 15a the left aircraft flies straight to AMS after which the top aircraft moves in to land. The right and bottom aircraft both move out of the way. A clear hierarchy is visible in the way the aircraft sequence. As soon as the first aircraft lands the next aligns its heading with AMS while the remaining aircraft keep out of the way. A holding pattern emerges in episode 22700 as shown in Figure 15d. The top aircraft would have violated separation criteria if it would have kept its heading, instead it flies a short holding pattern before successfully landing at AMS. Figure 15e displays a totally different strategy, despite failing. The two outer aircraft manoeuvre behind the two centre aircraft, but do so in a symmetric way causing them to collide into each other. In Figure 15f an phenomenon occurs that occurs frequently during training. The aircraft just misses AMS and ends up circling it before meeting the termination criteria.

After approximately 18000 episodes the aircraft sometimes start to show oscillatory behaviour by continuously overshooting the track angle to AMS in consecutive actions. The agent saturates the action space and is effectively only choosing a binary left or right.

E. Experiment C

1) Description: An attempt is made for agents to learn to generalize across multiple scenarios and expose the agent to a large variety of states. The previous experiment is repeated using an extended set of scenarios. This should improve exploration by exposing the model to a larger variety of scenarios within the same environment and possibly reduce overfitting. This may allow it to better generalize to situations it has not seen before.

The scenarios contain 4 types of formation in which the aircraft start: a triangle, a line of 3 aircraft, a line of 4 aircraft and a rectangle. The orientation and angles in these formation are randomized in each scenario as well as the scenario selection during training. This also means that in some of these scenarios a landing sequence is pre-embedded in the shape of the formation (e.g. the line) and in many of the scenarios a symmetric shape with limited manoeuvrability option must be sequenced and spaced (e.g. the rectangle).

The network is trained using 1000 different scenarios and will be evaluated on 100 scenario's it has never seen during training. These are 25 of each formation where the line formations do not form natural sequences.

2) Results: The learning curve for this experiment is shown in Figure 16. It shows a very chaotic progress of the learning process. Aircraft have learned to fly towards AMS in the multi-agent setting very well, however, there is a complete lack of coordination in the observed learned behaviour. The aircraft seem to be completely unaware of each others presence and show no signs of sequencing and spacing attempts. Rather, each aircraft flies straight towards AMS and as each aircraft converges to AMS they violate the separation minima. Feasible solutions where all aircraft land originate from scenarios that naturally form a landing sequence and don't require coordination if all aircraft follow a selfish policy as occurs for the rectangle formation in Figure 17.

F. Discussion

Three experiments have been performed to test a multi-agent setup. Experiment A showed that training with a shaped reward is more robust in this environment than a sparse reward. In this case it also did not change the optimal policy, because the optimal policy is still to align the heading of the aircraft with the bearing to Schiphol. In terms of the state space this means servo to $hdg_{qdr} = 0$. hdg_{qdr} is the third state available for observation for the aircraft. If $hdg_{qdr} = 0$ then this means the heading of the aircraft is aligned with the bearing to AMS. It is interesting to realize that for the single aircraft case this is similar to a control task where the error is hdg_{qdr} and the controller directly controls the aircraft target heading. The position of the aircraft becomes almost irrelevant to learn the optimal policy in the single aircraft case. Except that sensitivity to action outputs become larger nearby the airport. If the aircraft just misses the airport the aircraft may end up circling the airport.

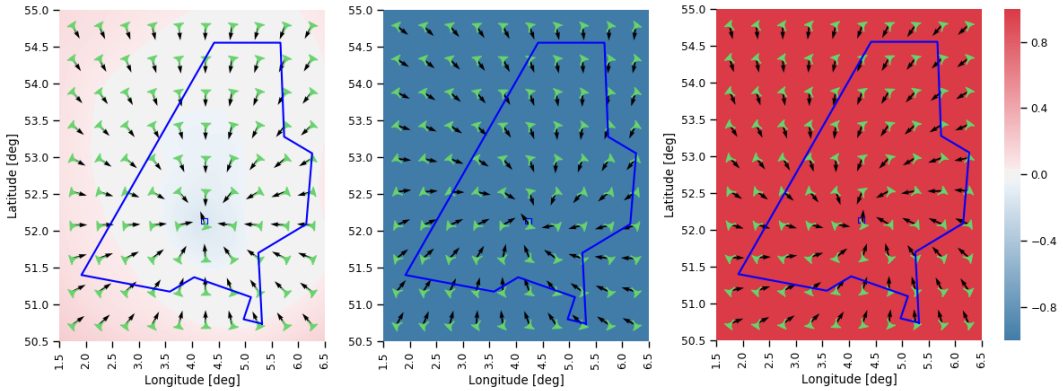


Fig. 13: Three heatmaps show action taken by aircraft over the state space. The left heatmap shows actions for $\mathrm{hdg}_{\mathrm{qdr}} = 0$, the middle heatmap for $\mathrm{hdg}_{\mathrm{qdr}} = 10$ and the right for $\mathrm{hdg}_{\mathrm{qdr}} = -10$. Action outputs are on the range [-1, 1]. The green aircraft show the aircraft position and heading and the black vectors show the aircraft target heading computed from the action output.

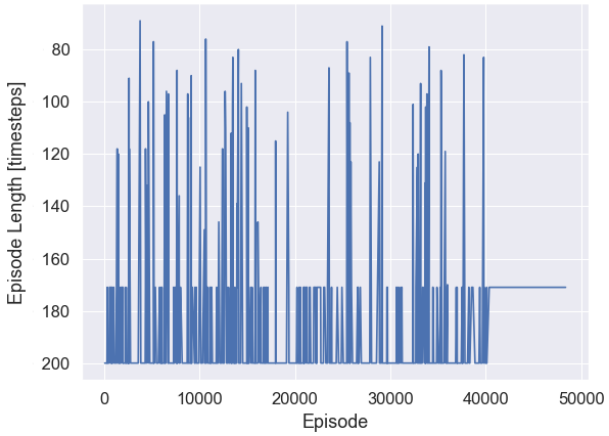


Fig. 14: Training progress for multi-agent training on a single scenario. If a loss of separation occurs episode length is set to 200, maximum episode length was 175 steps.

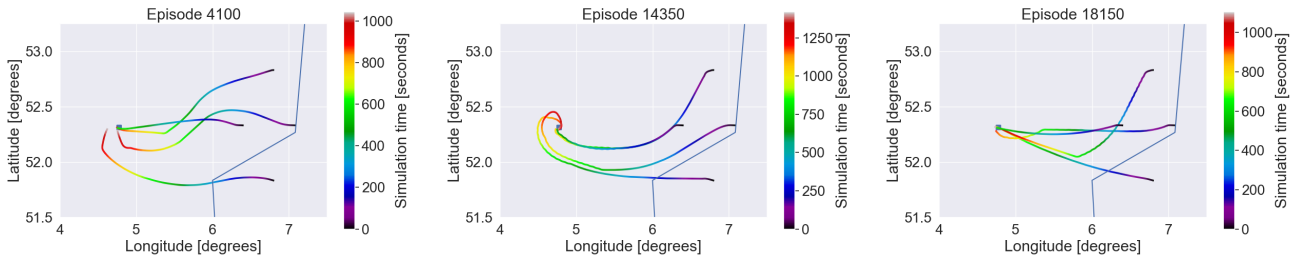
In experiment B BiCNet was unable to learn a optimal policy. For the multi-aircraft case the control problem analogy does not hold as hdg_{qdr} is no longer the only error that must be taken into account. The reward earned for flying in the direction of AMS is now contested by the additional loss of separation penalty. Therefore, in this case the addition of the shaped reward does change the optimal policy. Still, observing the trajectories flown by the aircraft do not show that aircraft are solely trying to follow $hdg_{qdr} = 0$. Instead aircraft do actively try to avoid each other. However, it could be that the policy simply learned a trick that only works on this scenario which does not generalize to unknown situations.

When attempting to find a policy that generalizes better over multiple scenarios in Experiment C, all aircraft started to ignore each others presence and started flying like in the single aircraft case. This inevitably leads to loss of separation. To find an explanation for this suboptimal behaviour the network weights are plotted. The weights of first hidden layer of the network are visualised in Figure 18. The row that lights up corresponds to the hdg_{qdr} state. Inspecting the weights in the first layer of the network shows that the agent heavily relies on heading information and much less so on the other states. This means that the information that passes on to the communication layer in the network is strongly composed of heading information. This suggests that it may be beneficial to use regularization techniques during network training to prevent action outputs from over relying on the aircraft heading. On the other hand it is true that the hdg_{qdr} state is the most important state when considering aircraft flying in a solo setting. This suggests that better solutions may be obtained if a shaped reward structure is designed that is not directly linked to a single state.

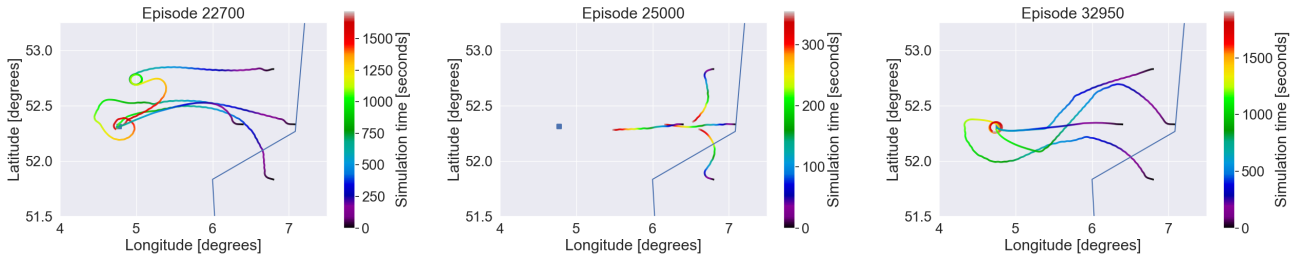
When comparing the setup of Experiment B with that of Experiment C the only difference are the scenarios used during training. Experiment B does show some emerging strategies where Experiment C only shows a selfish strategy and therefore fails to achieve the goal of the experiment. The failure of experiment C cannot be blamed completely on the reward function design. As also shown in Experiment A, a training run in reinforcement learning can succeed or fail based on a random seed set at initialization. All hyperparameters are equal, the traffic scenario is the same and the environment is identical. Yet, using a different random seed at the initialization of the experiment for the network weights and exploration noise can be the difference between finding an optimal policy and a failed training run.

V. Recommendations

Sequencing and spacing is a problem in 3D space, even though in this paper it has been simplified as a 2D problem where aircraft fly with constant speed. It actually becomes easier for aircraft to sequence and space when aircraft are



(a) 4100 episodes of training (b) 14350 episodes of training (c) 18150 episodes of training



(d) 22700 episodes of training (e) 25000 episodes of training (f) 32950 episodes of training

Fig. 15: Trajectories flown by the learned policy after training for the amount of episodes indicated. The color of the trajectory show the temporal progression of relative aircraft.

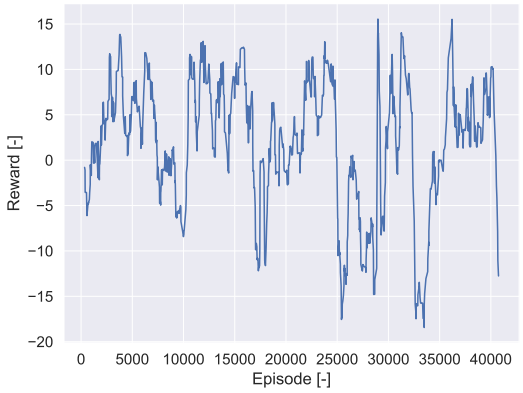


Fig. 16: Running average of 10 samples of the total reward received during and episode training for Experiment C.

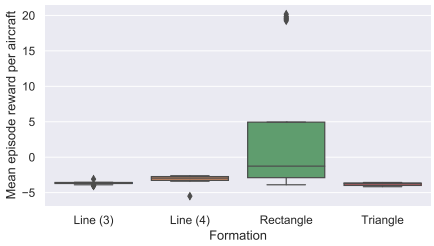


Fig. 17: Boxplot showing the test reward for 25 samples on each of the 4 formations in experiment C.

also allowed to make speed changes as it opens up a larger solution space. However, this does make the state-action space more complicated with the addition of an extra action. Therefore further research should focus on extending the environment to better represent the real world.

Another recommendation is to use shaped reward in a different form, because in the multi-agent environment as used in this paper it interferes with the optimal solution. This can for instance be done by setting intermediate objectives, e.g. setting up zones around the aircraft where an aircraft receives a reward when crossing from one zone to the other.

A way to not rely on the shaped reward is to utilize human demonstrations for the problem. Human demonstration can accelerate learning and provide safe exploration as shown by [25]. The human demonstration is not used to demonstrate the correct solution, because that would forfeit the purpose of searching for emerging strategies. Rather, it can be used as a means for efficient initial exploration. The reason the run with sparse reward failed in experiment A failed was that it never observed the reward for landing. This could be solved with human demonstrations.

VI. Conclusion

In the search for emerging strategies, multiple environment designs and reinforcement learning techniques were applied and assessed. A successful policy is found for learning to navigate under a 4D constraint for single aircraft using Duelling DQN. The aircraft arrived within 5 seconds of the STA while absorbing a delay in the range 0 to 100 seconds. The strategy it used is to fly a curved path to the FAF that somewhat resembles a path stretch manoeuvre. Delays assigned close to 0 seconds and 100 seconds failed. This was due to the discrete action space overshooting small delays. Additionally difficulties encountered with the Duelling-DQN

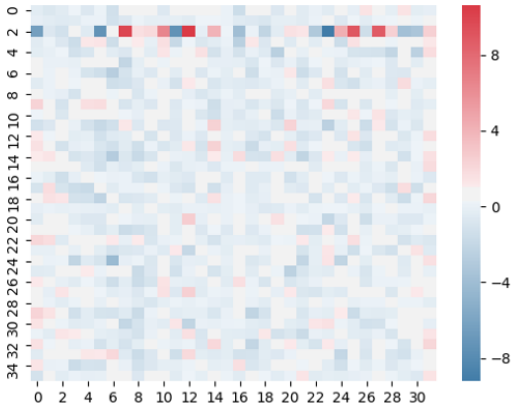


Fig. 18: Heatmap of the network weights of the layer after the concatenation of input states in the actor network. The vertical axis shows the input to the layer and horizontal actions shows the number of neurons in the layer itself.

algorithm and architecture include catastrophic forgetting as well as finding a good balance between exploration and exploitation. This made the algorithm and environment difficult to work with.

To address shortcomings and eliminate obstacles found in the experiment Duelling DQN, the 4D constraint is removed and a continuous action space was created to prevent oscillation artefacts due to the discrete action space. The DDPG algorithm with actor-critic formulation is implemented to train multiple agent collectively represented in a single policy network inspired by BiCNet. Training on a single scenario with 4 aircraft showed the emergence of a holding and emergence of priority among an aircraft sequence, but failed to converge to an optimal solution. Late in the training phase aircraft also started to oscillate, meaning they saturate the action space available. This effectively made the action space discrete and is a sign of overfitting.

Training on multiple scenarios using different scenarios with the aim to generalize better to unknown scenarios and reduce overfitting led to an absence of coordination. The shaped reward on the heading angle acted as a direct error signal and dominated the collective action inference and thereby preventing any coordination from emerging. The best improvement when continuing research on this environment design would be to improve the design of the reward function. The reward function should better reflect and encourage over-reliance on a single state.

References

[1] B. Favennec, E. Hoffman, A. Trzmiel, F. Vergne, and K. Zeghal, "The Point Merge Arrival Flow Integration Technique: Towards More Complex Environments and Advanced Continuous Descent," in *9th AIAA Aviation Technology, Integration, and Operations Conference (ATIO)*, 2009.

[2] L. Man, "An agent-based approach to automated merge 4D arrival trajectories in busy terminal maneuvering area," in *Procedia Engineering*, vol. 99, pp. 233–243, 2015.

[3] T. Nikoleris, H. Erzberger, R. A. Paielli, and Y.-C. Chu, "Autonomous System for Air Traffic Control in Terminal Airspace," in *14th AIAA Aviation Technology, Integration and Operations Conference*, (Atlanta), 2014.

[4] H. Erzberger, T. Nikoleris, R. A. Paielli, and Y.-C. Chu, "Algorithms for control of arrival and departure traffic in terminal airspace," in *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 230, pp. 1762–1779, 2016.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, feb 2015.

[6] H. v. Hasselt, "Double Q-learning," in *Proceedings of NIPS*, pp. 1–9, 2010.

[7] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," sep 2015.

[8] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv*, no. 9, pp. 1–16, 2016.

[9] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," nov 2015.

[10] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining Improvements in Deep Reinforcement Learning," 2017.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *ICLR 2016*, 2015.

[12] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, K. Kavukcuoglu, and K. Com, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1928–1937, PMLR, 2016.

[13] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," *NIPS'17 Proceedings of the 31th International Conference on Neural Information Processing SystemsProceedings*, pp. 6379—-6390, 2017.

[14] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning Multiagent Communication with Backpropagation," in *NIPS'16 Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2252–2260, 2016.

[15] P. Peng, Y. Wen, Y. Yang, Y. Quan, Z. Tang, H. Long, and J. Wang, "Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games," 2017.

[16] M. W. Brittain and P. Wei, "Towards Autonomous Air Traffic Control for Sequencing and Separation - A Deep Reinforcement Learning Approach," in *2018 Aviation Technology, Integration, and Operations Conference*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, jun 2018.

[17] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, University of Cambridge England, 1989.

[18] R. S. Sutton, D. Mcallester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *In Advances in Neural Information Processing Systems 12*, pp. 1057–1063, 1999.

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

[20] R. S. Sutton and A. G. Barto, *Reinforcement learning : an introduction*. MIT Press, 2nd ed., 2018.

[21] J. N. Tsitsiklis and B. V. Roy, "An Analysis of Temporal-Difference Learning with Function Approximation," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 42, no. 5, 1997.

[22] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION," *ICLR*, 2016.

[23] F. J. Pineda, "Generalisation of backpropagation to recurrent and higher order neural networks," *Physic Review Letter, 18*, pp. 2229–2232, 1987.

[24] J. M. Hoekstra and J. Ellerbroek, "BlueSky ATC Simulator Project: an Open Data and Open Source Approach," in *7th International Conference on Research in Air Transportation*, pp. 1–8, 2016.

[25] G. Schonebaum, J. Junell, and E.-J. Van Kampen, "Human Demonstrations for Fast and Safe Exploration in Reinforcement Learning," in *AIAA Information Systems-AIAA Infotech @ Aerospace*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, jan 2017.

APPENDIX

APPENDIX A

The algorithm for training Deep Q-Networks with experience replay is written down in algorithm 1. [5]

Algorithm 1: Deep Q-networks with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
for *episode=1, M* **do**
  Initialize sequence $s_1 = \{x_1\}$
  for *t=1,T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(s_t, a; \theta)$
    Execute action a_t in simulator and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
    if *episode terminates at step j+1* **then**
      Set $y_j = r_j$
    else
      Set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^-)$
    end
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every c steps set target network parameters $\theta^- \leftarrow \theta$
  end
end

The Deep Deterministic Policy Gradient Algorithm extended for multi-agent communication with BiCNet is shown in algorithm 2.

Algorithm 2: BiCNet algorithm [15]

Initialize actor network and critic network with ξ and θ
Initialize target actor network and target critic network with $\xi' \leftarrow \xi$ and $\theta' \leftarrow \theta$
Initialize replay memory R
for *episode=1, E* **do**
  Initialise a random process $\mathcal{U}$ for action exploration
  Receive initial observation state s^1
  for *t=1,T* **do**
    For each agent i, select and execute action $a_i^t = \mathbf{a}_{i,\theta}(s^t) + \mathcal{N}_t$
    Receive reward $[r_i^t]_{i=1}^N$ and observe new state s^{t+1}
    Store transition $\left\{s^t, [a_i^t, r_i^t]_{i=1}^N, s^{t+1}\right\}$ in R
    Sample random minibatch of transitions $\left\{s_m^t, [a_{m,i}^t, r_{m,i}^t]_{i=1}^N, s_m^{t+1}\right\}_{m=1}^M$ from R
    Compute target value for each agent in each transition using the Bi-RNN:
    for *m=1, M* **do**
      $\hat{Q}_{m,i} = r_{m,i} + \lambda Q_{m,i}^{\xi'}\left(s_m^{t+1}, \mathbf{a}_{\theta'}(s_m^{t+1})\right)$ for each agent i
    end
    Compute critic gradient estimation according to:
    $\Delta\xi = \frac{1}{M}\sum_{m=1}^M \sum_{i=1}^N \left[(\hat{Q}_{m,i} - Q_{m,i}^{\xi}(\mathbf{s}_m, \mathbf{a}_\theta(\mathbf{s}_m))) \cdot \nabla_\xi Q_{m,i}^{\xi}(\mathbf{s}_m, \mathbf{a}_\theta(\mathbf{s}_m))\right]$
    Compute actor gradient estimation:
    $\Delta\theta = \frac{1}{M}\sum_{m=1}^M \sum_{i=1}^N \sum_{j=1}^N \left[\nabla_\theta \mathbf{a}_{j,\theta}(\mathbf{s}_m) \cdot \nabla_{\mathbf{a}_j} Q_{m,i}^{\xi}(\mathbf{s}_m, \mathbf{a}_\theta(\mathbf{s}_m))\right]$
    and replace Q-value with the critic estimation
    Update the networks based on Adam using the above gradient estimators
    Update target networks:
    $\xi' \leftarrow \gamma\xi + (1-\gamma)\xi'$, $\theta' \leftarrow \gamma\theta + (1-\gamma)\theta'$
  end
end

Part II

Preliminary Thesis

Faculty of Aerospace Engineering

Towards Emerging Sequencing and Spacing Strategies for a self-learning ATC

Preliminary Thesis

Bart Vonk

18 November 2018

TU Delft Delft University of Technology

Challenge the future

Towards Emerging Sequencing and Spacing Strategies for a self-learning ATC

Preliminary Thesis

Preliminary Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

Bart Vonk

18 November 2018

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

Delft University Of Technology
Department Of
Control and Simulation

Dated: 18 November 2018

Readers: _____

Acronyms

A3C	Asynchronous Advantage Actor-Critic
AMAN	Arrival Manager
ATC	Air Traffic Control
ATCo	Air Traffic Controller
ATM	Air Traffic Management
Bi-RNN	Bidirectional Recurrent Neural Network
BiCNet	Bidirectional Communication Network
C&S	Control & Simulation
CAS	Calibrated Airspeed
CD&R	Conflict Detection & Resolution
CommNet	Communication Network
DDPG	Deep Deterministic Policy Gradient
DDRII	Demand Data Repository II
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
FAF	Final Approach Fix
FIR	Flight Information Region
GAE	Generalized Advantage Estimate
IAF	Initial Approach Fix
KL	Kullback-Leibler
LSTM	Long Short Term Memory
MacDEC-POMDP	Macro Decentralized Partially Observable Markov Decision Process
MADDPG	Multi Agent Deep Deterministic Policy Gradients
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
NLP	Natural Language Processing
OU	Ornstein-Uhlenbeck
PM	Point Merge
PPO	Proximal Policy Optimization
PS-TRPO	Paramater Sharing Trust Region Policy Optimization

ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SARSA	State Action Reward State Action
SESAR	Single European Sky ATM Research
SID	Standard Instrument Departure
SMDP	Semi-Markov Decision Process
STA	Scheduled Time of Arrival
STAR	Standard Terminal Arrival Route
TAR	Terminal Auto Resolver
TMA	Terminal Manoeuvring Area
TRPO	Trust Region Policy Optimization

List of Symbols

Greek Symbols

α	Learning rate
δ	Parameter update values for actor-critic
ϵ	Probability of taking a random action
ϵ	clipping parameter for gradient clipping
γ	Discount factor for cumulative rewards
λ	Tuning parameter for General Advantage Estimation
λ	Learning rate
μ	Deterministic policy
π	(Stochastic) Policy
ψ	Heading
ρ	State distribution
τ	Discount factor adjusted for the duration of an action
θ	Parameters of a parameterized policy
ξ	Parameters of actor network

Roman Symbols

A	Advantage
$\mathcal{A}$	Set of actions for a Markov Decision Process
a	Action
a	Hidden state in recurrent neural network
b	Bias
c	Target network reset factor

D	Replay memory
d	Static state distribution
d	Distance from target to aircraft
$\mathbb{E}$	Expected value
e	Transition tuple
$g(\cdot)$	Activation function
J	Objective function
$\mathcal{L}$	Loss function
$\mathcal{P}$	Set of transition probabilities for a Markov Decision Process
Q	Q-Value or Q-function
$\mathcal{R}$	Set of rewards for a Markov Decision Process
r	Reward
$\mathcal{S}$	Set of states for a Markov Decision Process
s	State
T	Sequence Length
t	Time step
$\mathcal{U}$	Random process
V	Value or Value function
W	Weight matrix for a neural network
x	Sequence input for a Recurrent Neural Network
y	Sequence output for a Recurrent Neural Network

Subscripts

a	When performing action
avg	Average
max	Maximum value
t	At time step

Superscripts

$*$	Optimal
$'$	At next timestep
$\hat{}$	Target function
$-$	Parameters of the target network
π	When executing a policy
$<>$	Sequence number

Contents

Acronyms v

List of Symbols vii

List of Figures xiv

List of Tables xv

1 Introduction 1

2 Research question, aims and objectives 3

3 Literature 5

3-1 Introduction to reinforcement learning . . . 5

3-1-1 Markov decision processes . . . 5

3-1-2 Semi-Markov decision process . . . 7

3-1-3 Example: Nchain environment . . . 7

3-1-4 Policy gradient methods . . . 8

3-1-5 Deterministic policy gradients . . . 9

3-1-6 Q-learning . . . 10

3-1-7 SARSA . . . 11

3-1-8 The exploration versus exploitation dilemma . . . 11

3-1-9 Actor-critic methods . . . 12

3-1-10 Hierarchical reinforcement learning . . . 14

3-1-11 Learning aids . . . 14

3-2 Deep reinforcement learning . . . 14

3-2-1 Deep Q-networks . . . 14

3-2-2 Double deep Q-networks . . . 15

3-2-3 Dueling deep Q-networks . . . 16

3-2-4 Deep deterministic policy gradients . . . 16
3-2-5 Asynchronous advantage actor-critic . . . 17
3-3 Multi-agent reinforcement learning without communication . . . 17
3-3-1 Learning type settings . . . 18
3-3-2 Challenges in multi-agent reinforcement learning . . . 18
3-3-3 Multi-agent deep deterministic policy gradients . . . 18
3-3-4 Event-driven multi-agent reinforcement learning . . . 19
3-3-5 Proximal policy optimization . . . 19
3-4 Multi-agent reinforcement learning with communication . . . 20
3-4-1 Sequence data modelling . . . 20
3-4-2 BiCNet . . . 21
3-4-3 CommNet . . . 22
3-4-4 Algorithm Overview . . . 22
3-5 Automation in TMA for ATC . . . 23
3-5-1 Terminal AutoResolver . . . 23
3-5-2 Point merge system . . . 25
3-5-3 Sequencing and spacing with genetic algorithms . . . 25
3-6 Application of reinforcement learning in ATM . . . 25

4 Preliminary Analysis **27**
4-1 Conflict resolution using MADDPG . . . 27
4-1-1 Experiment description . . . 27
4-1-2 Method . . . 27
4-1-3 Results . . . 28
4-2 4D trajectory navigation with DQN and Dueling DQN . . . 28
4-3 Experiment description . . . 29
4-4 Method . . . 30
4-4-1 Results for DQN . . . 31
4-4-2 Results for Duelling DQN . . . 32
4-4-3 Discussion . . . 34
4-5 Navigation by waypoint selection . . . 36
4-6 Experiment description . . . 37
4-7 Method . . . 37
4-8 Results and Discussion . . . 38
4-9 Conclusion . . . 40

5 Thesis proposal **43**
5-1 Problem statement . . . 43
5-1-1 Generating traffic scenarios . . . 44
5-2 Method . . . 45
5-3 Verification & Validation . . . 45
5-4 Outcome . . . 47
5-5 Planning . . . 48

6 Conclusion **51**

A DQN Summary **53**

B Dueling DQN summary **55**

Bibliography **59**

List of Figures

3-1 Block diagram of a reinforcement learning process 6
3-2 Nchain environment diagram showing state transitions, actions and rewards 8
3-3 Schematic of Actor-Critic Architecture 13
3-4 Dueling DQN architecture 16
3-5 MADDPG training vs execution phase 19
3-6 Unrolled recurrent neural network 21
3-7 BiCNet architecture 22
3-8 CommNet architecture 23
3-9 Schematic overview of the Point Merge System Man (2015) 25

4-1 Coordinated turning manoeuvre by 3 aircraft observed when training with MADDPG . . . 28
4-2 Component break down of the problem representation 28
4-3 Airspace layout for problem design 29
4-4 State representation 31
4-5 Action Space 31
4-6 Distance vs. time for 25 test episodes after 1000 episodes of training with DQN 33
4-7 t_{delay} vs time for 25 test episodes after 1000 episodes of training with DQN 33
4-8 Reward vs. time for 25 test episodes after 1000 episodes of training with DQN 33
4-9 Trajectories for 25 test episodes after 1000 episodes of training with DQN 33
4-10 t_{delay} at termination for 25 test episodes after 1000 episodes of training with DQN 34
4-11 Running average of 10 samples for the final reward received during an episode while training with DQN 34
4-12 Distance vs. time for 25 test episodes after 375 episodes of training with dueling DDQN . 34
4-13 t_{delay} vs time for 25 test episodes after 375 episodes of training with dueling DQN 34

4-14 Reward vs. time for 25 test episodes after 375 episodes of training with dueling DQN . . . 35
4-15 Trajectories for 25 test episodes after 375 episodes of training with dueling DQN 35
4-16 t_{delay} at termination for 25 test episodes after 375 episodes of training with dueling DQN 35
4-17 Running average of 10 samples for the final reward received during an episode while training with dueling DQN 35
4-18 Cross track navigation with waypoint 36
4-19 Network architecture of the BiCNet implementation in BlueSky to control air traffic simulation 39
4-20 Network architecture of the BiCNet implementation in BlueSky to control air traffic simulation 41

5-1 Heading range for action selection 44
5-2 Network architecture of the BiCNet implementation in BlueSky to control air traffic simulation 46
5-3 Thesis planning 49

List of Tables

3-1 Example value table for Nchain environment at initialization 9

3-2 Q-table for Nchain environment after learning 11

3-3 Overview and classification of reinforcement learning algorithms. D is for Discrete and C is for Continuous 23

3-4 Confict resolution manoeuvres 24

4-1 Reward function components 30

4-2 Hyperparameter overview 32

5-1 RECAN-EU minimum wake separation criteria during approach 45

A-1 Reward function parameters 53

A-2 DQN network summary 53

A-3 Hyperparameter overview 53

B-1 Reward function parameters 55

B-2 Dueling DQN network summary 55

B-3 Hyperparameter overview 56

Chapter 1

Introduction

For en-route air traffic, 4D-trajectory based operation provides a way to automate the task of an air traffic controller. However, in the terminal area closer to the airport the combination of interaction and unpredictability still requires tactical decisions. The unpredictability originates from departing pop-up flights and multiple air traffic flows merging into landing sequences. In addition, lower airspace is subject to environmental restrictions due to noise and emissions. Air travelling has become a commodity due to cheap tickets fairs, increasing the demand for air traffic. [1] The increased amount of air traffic also leads to more incidents. Minimum separation violations have increased in Dutch air space, most of which occurred in the flight phase just before or during approach [2].

The Terminal Manoeuvring Area (TMA) is a complex airspace. Currently, in approaches to Schiphol aircraft are procedurally separated by following a Standard Terminal Arrival Route (STAR) or a Standard Instrument Departure (SID). Aircraft are placed in holding stacks before they are cleared to proceed to the runway. Then arriving aircraft are guided to the runway by Air Traffic Control (ATC) vectors for the last part of the flight. Placing aircraft in holding stacks is very inefficient as aircraft are flying extra distance while flying in circles, which is not fuel efficient. Improvements have been made by recent attempt as part of Single European Sky ATM Research (SESAR) with the new Point Merge (PM) system, which allows the implementation of continuous descend approaches under high traffic loads. The PM system is already in use in Paris and Oslo. New sequencing and spacing techniques are still continuously researched [3].

Air traffic controllers are constrained in the way they work by the procedures in which they operate. The backbone of the arrival and departure system around Schiphol are the SIDs and STARs with sequencing and spacing. New structures are being investigated, like the Point-Merge system. Engineering a system like PM is essentially designing the strategy for the sequencing and spacing of aircraft. Could it instead be possible to have a computer learn how to efficiently solve the sequencing and spacing problem within constraints drawn from safety criteria and workload to learn from the emerging strategies? Recent developments in Artificial Intelligence have shown emerging strategies in complex video games which surpass human level performance. Initial results range from relatively

[1] Annual Report Schiphol Group 2017

[2] Annual Report LVNL 2016

[3] Eric Hoffman, 'Point Merge: Improving and Harmonising Arrival Operations', 2016, url: `https://www.eurocontrol.int/services/point-merge-concept` [accessed: July 6, 2018]

simple games like breakout Mnih et al. (2013) to most recently very complex games that take humans years to master like Dota 2 [4]. This work explores the possibilities to apply these techniques to the sequencing and spacing of aircraft in the TMA and work towards discovery of more efficient and safe operation strategies in the TMA.

The layout of this work is as follows. First the research question aims are presented in chapter 2. After the objective of the research is clear relevant literature on the subject is discussed in chapter 3. A preliminary analysis has already been performed based on the literature. This can be found in 4. Finally a proposal for the final thesis research is done in chapter 5 followed by the conclusions in chapter 6.

[4]OpenAI research team, 'OpenAI Five', 2018, url: `https://blog.openai.com/openai-five/`, [accessed: June 25, 2018]

Chapter 2

Research question, aims and objectives

The objective of this research is to find emerging strategies for air traffic control in the TMA by applying reinforcement learning to the sequencing and spacing problem and identify the emerging strategies. This will be done while answering the research questions: What is the safety and efficiency of the emerging strategies for sequencing and spacing aircraft in the TMA, learned by a controller trained using reinforcement learning?

The following subquestions can be defined:

1. What reinforcement learning techniques are suitable to learn the sequencing and spacing tasks in the terminal area?
2. What emerging strategies can be identified when looking at conflict resolution, flown trajectories and sequence order?
3. What is the safety and efficiency of the controllers actions in terms of fuel, delay and controller robustness to alternate scenarios?

The following subgoals have to be reached to answer the research questions:

1. Study reinforcement learning techniques.
2. Model the environment and agents in terms of state observations and action sets.
3. Implement a reinforcement learning algorithm in BlueSky.
4. Create traffic scenarios for training and validation.
5. Have the controller learn by an iterative process of training and tuning the hyper-parameters.
6. Identify emerging control strategies.
7. Evaluate the safety and efficiency and robustness of the learned controller.

Chapter 3

Literature

Machine learning can be divided into three categories: supervised learning, unsupervised learning and reinforcement learning. In this literature study only reinforcement learning is studied. Reinforcement learning is a suitable method to apply to the research problem, because it requires no supervision during learning. Time is also important in reinforcement learning as opposed to unsupervised learning where data points are not time dependent. Reinforcement learning is capable of learning sequential decision making and is therefore suitable for the task.

The literature is split into two parts. The first part is concerned with the technical aspects of reinforcement learning. The second part will deal with automation of air traffic management.

3-1 Introduction to reinforcement learning

Reinforcement learning is a machine learning technique in which an agent tries to optimize the reward it receives for performing actions in an environment. The basic block diagram that describes a reinforcement learning process is shown in Figure 3-1. At each timestep the agent chooses an action from its policy. The agent interacts with the environment by executing the action, which changes the state of the environment. It then observes the reward it receives and the new state of the environment. The agent's policy is thus a mapping from state to action. The controller can then optimize its policy based on the reward the environment returns to maximize the sum of the future rewards received from the environment. Since the agent is optimizing its actions such that it receives high reward, the foundation for the desired behaviour of an agent lies in the reward function. The reward can be seen as a learning signal and is often compared with how humans learn. If an action leads to a high reward, this behaviour is reinforced as opposed to when an action leads to a punishment. The reward function should steer the agent to the desired result, but does not necessarily specify how the agent should achieve this result. In soccer one is rewarded for scoring the goal, however, there are multiple ways of doing this. The environment with which the agent interacts can either be episodic or have an infinite time horizon. The environment is reset when a termination conditions is met, or when the agent has achieved the goal. This is also referred to as an episode.

3-1-1 Markov decision processes

Reinforcement learning is based on a Markov Decision Process (MDP) Sutton and Barto (1998). A MDP is a form of a sequential decision model. The state transition of a MDP satisfies the Markov

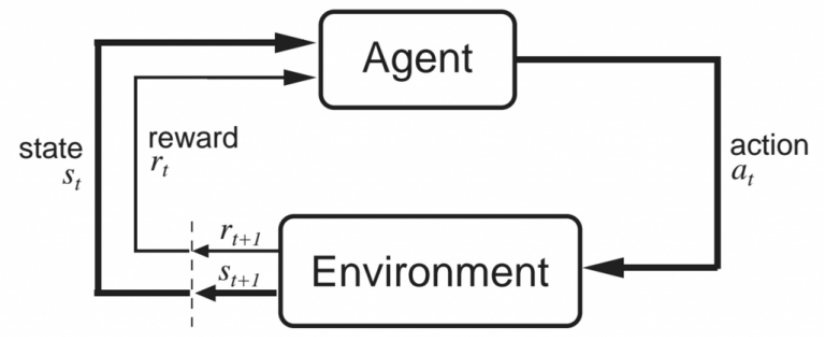


Figure 3-1: Block diagram of a reinforcement learning process

Property that the transition is independent of previous states and actions and thus only depends on the current state s and action a. A MDP can be represented by the tuple ($\mathcal{S}$, $\mathcal{A}$, P_a, R_a, γ) where:

- $\mathcal{S}$ is the set of states.
- $\mathcal{A}$ is the set of actions.
- $\mathcal{P}_a$ is the transition probability that action a in state s will lead to state s' at that timestep.
- R_a is the immediate reward received by the agent for the state transition from s to s' by performing action a.
- γ is the discount factor, which represents the relative importance of future and immediate rewards.

At every timestep t an agent will select an action a from $\mathcal{A}$ using policy π. The goal is to find an optimal policy π^* to maximize the total future reward. In order to choose the right actions a prediction of future rewards has to be made. The discount factor $\gamma \in [0, 1]$ represents the relative value of immediate rewards with respect to delayed rewards. The total discounted reward R_t is computed using the discount factor as in Equation 3-1. Rewards are discounted to avoid infinite returns for MDPs with infinite time horizons. A second reason to discount future rewards is the growing uncertainty of obtaining the rewards and the value of the rewards when predicting further into the future. There are two equations that help predicting the total future discounted reward. The first is the value function V^π, which represents the expected future reward in the current state when following policy π, where π is modelled as a probability distribution over the actions $\mathcal{A}$. See Equation 3-2. The second is the Q-function Q^π, which represents the expected reward for state-action pairs for a policy π. See Equation 3-3. It is important to note that both V and Q are strictly linked to a policy π. They return different values for the same state when following a different policy. The intuition behind this is that the behaviour of the agent is inherently different when following another policy, hence being in a state may be more valuable depending on the behaviour a policy prescribes. The definition of V and Q also use the expectation operator to deal with the randomness in the process as a policy and/or the environment may be stochastic.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3-1}$$

$$V^\pi(s) = \mathbb{E}_\pi(R_t | s_t = s) \tag{3-2}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi(R_t | s_t = s, a_t = a) \tag{3-3}$$

In order to compute the values for the value function and the Q-function the Bellman equations are used. They are derived by substituting the expected value for the discounted return R_t as $R^a_{ss'}$ into the value functions and using transition probability $P^a_{ss'}$ to manipulate them into a recursive form as shown in Equations 3-4 3-5. The recursive form is of fundamental importance to solving MDPs as it allows to express the value of a state as the value of other states. This allows for iterative calculations (temporal difference) of Q-values, which is the foundation for the algorithms used to solve reinforcement learning problems.

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')] \tag{3-4}$$

$$Q^\pi(s,a) = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \sum_{a'} Q^\pi(s',a')] \tag{3-5}$$

3-1-2 Semi-Markov decision process

In a MDP there is the underlying assumption that each state transitions consumes the same amount of time. However, in some situations this is not a realistic assumption. In a Semi-Markov Decision Process (SMDP) a generalisation is made to allow for temporally extended actions Sutton, Precup, and Singh (1999). These actions can be considered higher-level policies or macro-actions with arbitrary length. The transition time can thus be modelled as a random variable. Consequentially the value function definition for discounted reward changes to Equation 3-6 and the Q-function update function changes to Equation 3-7. The parameter τ represents the possible duration of the temporally extended action. The discount factor is now adjusted to the actual time the action took and the rewards that were accumulated during the temporally extended action are also incorporated. In this way the rewards are accredited to the agent over a consistent time domain.

Take for example a normal MDP where an action could be to take a step left or a step right. This action will always consume the same amount of time. In a acSMDP the equivalent would be to walk towards a landmark. The duration of the action then depends on the relative position to the landmark.

$$V^\pi(s) = r(s,a) + \sum_{s'} \gamma^\tau P(s', \tau | s, a) V_\pi(s') \tag{3-6}$$

$$Q_{k+1}(s,a) = Q_k(s,a) + \alpha_k \left[r_{t+1} + \gamma r_{t+2} + ... + \gamma^{\tau-1} r_{t+\tau} + \gamma^\tau \max Q_k(s',a') \right] \tag{3-7}$$

3-1-3 Example: Nchain environment

To illustrate the above a practical example is given. A game is being played called Nchain available in OpenAI gym Brockman et al. (2016). Nchain is a simple game with a state set $\mathcal{S}$ consisting of 5 states shown in Figure 3-2. The player starts at state 0 and can perform two actions from action set $\mathcal{A}$. Action 0 means moving forward to the next state, and action 1 means moving back to state 0. The agent receives 0 reward for moving forward in all states except in state 4, where it will receive 10 reward for moving forward. The agent will always receive 2 reward for choosing action 1. Additionally there is a small probability that the action chosen by the agent is flipped by the environment, which is the transition probability $\mathcal{P}_a$. There is a clear distinction between the immediate reward received and a delayed reward, which can be earned by moving forward long enough without earning any reward in intermediate step.

The problem may be tackled in many ways. A very simple approach greedy approach where choosing the action with highest reward in each state will not yield the optimal policy. for which it receives 0 reward or it can be return to state 0, for which it receive 2 reward. Only when the agent is in state 4 it will receive 10 reward for moving forward, in which case it will remain in state 4. The agent has to

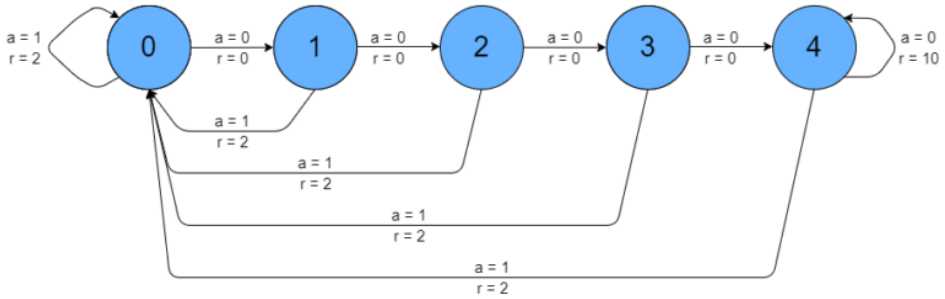


Figure 3-2: Nchain environment diagram showing state transitions, actions and rewards [1]

choose between immediate reward and a higher delayed reward obtained after reaching state 4. The agent can learn to deal with a delayed reward by making use of the discount factor to discount future rewards. Thus the value of the discount factor controls to what extend the agent uses possible future rewards to make a decision about the actions used. These will be illustrated in the following sections when methods for solving reinforcement learning problems are explained.

3-1-4 Policy gradient methods

The formulation of a reinforcement learning problem is now clear. The next step is to find an optimal policy. Policy gradient methods directly try to find an optimal policy by performing gradient ascent on the policy to maximize the expected return Sutton and Barto (1998). The gradients are calculated using a objective function, which can be considered the equivalent of the cost function in a optimization process. Therefore it is a on-policy algorithm, because it bootstraps gradients calculated by evaluating a policy to update the same policy. Policy gradient methods are guaranteed to converge to at least a local minimum. Again this may also be a disadvantage as it may be hard to find a global optimum. They can handle both discrete and continuous states and actions. Policy gradient methods make use of a parametrized policy $\pi(\theta)$. The policy is updated with the update rule $\theta = \theta - \Delta\theta$. In order to find the gradients an objective function $J(\theta)$ is specified, which is then maximized. Example objectives are start value of the value function of each episode, average value over an episode or the average reward per timestep. So eg. when optimizing for the start value objective function, the policy will be optimized for the highest start value of an episode.

$$\begin{aligned} J_1(\theta) &= V^{\pi_\theta}(s1) = \mathop{\mathbb{E}}_{\pi_\theta}[V_1] \\ J_{avgV}(\theta) &= \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s) \\ J_{avgR}(\theta) &= \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a) R_s^a \end{aligned} \tag{3-8}$$

d^{π_θ} in the objective functions is the static state distribution. The static state distribution can be seen as the distribution of visited states when the amount of steps taken in the environment goes to infinity and the distribution of value over the states does not change any more. This can be rewritten as the expected value of the return, which is what is being optimized with reinforcement learning. The policy gradients are then defined as $\nabla_\theta J(\theta)$ and $\Delta\theta = \alpha \nabla_\theta J(\theta)$ with step size (learning rate) α.

[1] Andy, 'Adventures in Machine Learning - Reinforcement learning tutorial using Python and Keras' [BLOG] `http://adventuresinmachinelearning.com/reinforcement-learning-tutorial-python-keras/`, visited 27-09-2018

Table 3-1: Example value table for Nchain environment at initialization

State	0	1	2	3	4
Value $V^{\pi}(s)$	0	0	0	0	0

The first are finite-difference methods that use finite-difference between two transitions to estimate the policy gradients. This stochastic optimization introduced a lot of variance but low bias. Another well-known policy gradient algorithm is REINFORCE Williams (1992), which uses episodic updates to measure the discounted return R_t for the entire episode to compute the state-action value for the policy gradient calculation. Therefore it can be considered a Monte Carlo method as well. In general the policy gradient methods tend to suffer from high variance because the direct updates of the policy can reinforce good action, but also weaken them. Imagine when an agent is at a three forked road, of which all actions yield negative reward. Choosing the action with the highest reward still leads to discouraging of choosing that action even though it was the best choice in that situation. This leads to a very noisy update signal.

Gradients may also be computed analytically rather than relying on sampling. This can be done using likelihood ratios. The likelihood ratio method exploits the following property of likelihood ratios as shown in Equation 3-9. Then using the policy gradient theorem as derived by Sutton, Mcallester, Singh, and Mansour (1999) the policy gradients can be computed analytically. The policy gradient theorem states that for any differentiable policy $\pi_\theta(s,a)$ the policy gradient is as shown in Equation 3-10

$$\begin{aligned}\nabla_\theta \pi_\theta(s,a) &= \pi_\theta(s,a)\frac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} \\ &= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\end{aligned} \tag{3-9}$$

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right] \tag{3-10}$$

As an example the policy gradients for the REINFORCE algorithm are shown for the Nchain environment. The policy π is a function with parameters θ and objective function $J_{avgV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$ will be used to optimize. The objective function can therefore also be seen as the cost function for the optimization process. The Nchain environment has a discrete state space, therefore the value function can be represented by as a value table with an entry for every state. As the agent samples transitions from the environment and observes the reward, the value table can be updated using the discounted reward as defined in Equation 3-6. Over time when all states have been visited a sufficient amount of times the value function represents the value of a state. It does not only say something about the immediate reward that can be earned when transitioning to that state, but it also reflects the opportunity that state offers to earn higher rewards in the future. The value of states is part of the objective function and thus used to deliver a gradient to the policy. The stochastic policy is simply a function of the state that outputs a probability distribution over the actions, from which the action can be sampled. So for the Nchain environment one can imagine that the value of state 4 is much higher than that state 0.

3-1-5 Deterministic policy gradients

Traditional policy gradient method rely on sampling a stochastic policy to compute the policy gradient and update the policy parameters using gradient ascent. Silver et al. (2014) have proven that a Deterministic Policy Gradient (DPG) exists. Instead of using a stochastic policy $\pi(a|s)$ the action is selected deterministically according to a deterministic policy $\mu(s)$. Analogous to the stochastic

policy gradient the deterministic policy gradient methods use an objective function as shown in Equation 3-11, where ρ^{μ} is the state distribution. The deterministic policy gradient theorem as derived by Silver et al. (2014) then provides that the gradient can be calculated by Equation 3-12.

They also proved that the deterministic policy gradient is a limiting case of the stochastic policy gradient when the policy variance goes to zero. For stochastic policies, the policy gradients are integrated over the action and state space, where the deterministic policy gradient are integrated only over the state space. This means that the environment can be sampled much more efficiently. The fact that the deterministic policy gradient is in fact a limiting case of the stochastic policy gradient means that the methods used in policy gradient can also be applied to deterministic policy gradients.

To illustrate the difference between a stochastic policy and a deterministic in practice, the following illustration is given. A stochastic policy learns a probability distribution for its actions. Say, it uses a Guassian to sample its actions from. Since the deterministic policy is a limiting case of the stochastic policy with zero variance, the deterministic policy directly outputs the action without the need to sample. One issue with DPG is that when there is not sufficient noise in the environment exploration will be poor. This can be countered by adding noise to the DPG, but this makes it stochastic again. Therefore DPG is often used with off-policy actor-critic algorithms to learn a deterministic target policy from trajectories that have been been generated by a stochastic policy. Actor-critic methods will be discussed later in subsection 3-1-9

$$\begin{aligned} J\mu_{\theta} &= \int_{S} \rho^{\mu}(s) r(s, \mu_{\theta}(s))\, ds \\ &= \mathbb{E}_{s\sim\rho^{u}}[r(s, \mu_{\theta}(s))] \end{aligned} \tag{3-11}$$

$$\begin{aligned} \nabla_{\theta} J(\mu_{\theta} &= \int_{S} \rho^{\mu}(s) \nabla_{\theta}\mu_{\theta}(s) \nabla_{a} Q^{\mu}(s,a)|_{a=\mu_{\theta}(s)}\, ds \\ &= \mathbb{E}_{s\sim\rho^{u}}[\nabla_{\theta}\mu_{\theta}(s) \nabla_{a} Q^{\mu}(s,a)|_{a=\mu_{\theta}(s)}] \end{aligned} \tag{3-12}$$

3-1-6 Q-learning

Previously described methods were all based on calculating the gradients of the policy directly. Q-learning is a value based method, which means it uses values to learn a policy. More specifically, Q-learning utilizes the Q-function as defined in Section 3-1-1 to choose actions that will yield the highest reward at termination of the episode Watkins (1989). It is a model-free approach, which means it directly tries to learn an optimal policy by interacting with the environment, rather than trying to learn transition and reward models for the environment. The policy in Q-learning can be represented as taking a greedy approach as $\pi(s) = \arg\max Q(s,a)$ for a discrete action set. Choosing the action with the highest Q-value thus results in the highest total reward if an optimal Q-function Q^{*} can be found. Using the Bellman equation in Equation 3-5, the Q-function can iteratively be updated by interacting with the environment to learn Q^{*} by using the update rule in Equation 3-13 with learning rate α. The tuple (s, a, r, s') is obtained by performing action a in state s and observing reward r and next state s'. The error to update Q is computed by taking the maximum Q-value in the next state s' and subtracting it from the Q-value of the action. It is thus an off-policy algorithm, because it uses the Q-value of the greedy action in the next state s' to update the current Q-values. The Q-value represents the expected discounted future reward so this update takes the Q-function closer to the real Q-function. In order to learn the correct Q-values for all states the agent must visit these states a sufficient number of times to learn the Q-values. Therefore the agent must explore the

Table 3-2: Q-table for Nchain environment after learning [2]

State	0	1	2	3	4
Action 0	62.74	66.32	70.83	76.64	84.51
Action 1	61.35	62.27	63.26	64.76	66.50

environment sufficiently during training. For discrete state spaces, the Q-function can be represented by a table with Q-values for each discrete state. This is also called the Q-table.

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max Q(s',a') - Q(s,a)) \tag{3-13}$$

A practical example using the Nchain environment. In Q-learning the Q-values are learned, which are state-actions pairs. Again, a table can be used to represent the Q-values for discrete state-actions pairs. For the Nchain environment this means two values have to be learned for every state, because for each state there are two possible actions. The Q-table is used by the agent to choose an action. It will choose the action with the highest Q-value, because the Q-value represents the highest expected future discounted reward. The Q-values in Table 3-2 are the result of applying Q-learning to the Nchain environment. As expected the agent learned that choosing action 0 yields a higher future discounted reward. Also the value of the progressive states show that the value of the states are higher, which is expected due to the delayed reward that can be obtained in state 4.

3-1-7 SARSA

Just like Q-learning SARSA is also a value based method. State Action Reward State Action (SARSA) is an abbreviation for the tuple S (s, a, r, s', a') Rummery and Niranjan (1994) Singh and Sutton (1996). The agent interacts with the environment by choosing an action a in state s from policy π. It will then observe the reward r and the next state s' and use the same action a' to estimate the error for the Q-value. The Q-value is then updated with learning rate α as shown in Equation 3-14. The error by inferring the same policy as for the action selection is computed using the same policy, because the same action is used in the next state s'. Hence, SARSA is an on-policy algorithm. The SARSA algorithm approaches the optimal policy during exploration, because it learns a sub-optimal policy. This means the policy receives penalties for wrong exploration moves as well. Due to this aspect SARSA can be viewed upon as a more conservative algorithm than Q-learning as it is less likely to take actions that yield negative reward during exploration. An illustration for this given in Sutton and Barto (1998) is an cliff-walking agent. Q-learning will directly learn the optimal policy by falling of the cliff many times due to the nature of random off-policy exploration where SARSA will more likely avoid the cliff because the actions are directly discouraged by updating the current policy.

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a)) \tag{3-14}$$

3-1-8 The exploration versus exploitation dilemma

The term exploration and the importance of exploration for the learning process have been highlighted a number of times already. This subsection discusses the concept in more depth. When an agent is learning to maximize a reward obtained from an environment it has to explore the environment around him. More specifically when taking Q-learning as an example, the agent has to learn the

[2] Andy, 'Adventures in Machine Learning - Reinforcement learning tutorial using Python and Keras' [BLOG] `http://adventuresinmachinelearning.com/reinforcement-learning-tutorial-python-keras/`, visited 27-09-2018

Value function for each state or the Q-function for each state-action pair. To do this the agent must visit these states or state-action pairs. This is called exploration. Once the agent has learned enough about the environment it can start to exploit what it has learned by following the learned policy. The question when and how to switch from exploration to exploitation is also known as the exploration versus exploitation problem in reinforcement learning.

Numerous strategies exist to explore the environment for discrete action space:

- ϵ-*greedy* policy. That is with probability ϵ a random action is choosen using a uniform distribution over the action set $\mathcal{A}$. If no random action is selected the agent will select the action with a greedy approach by choosing an action with the highest expected reward. ϵ is annealed over time to gradually move from pure exploring to pure exploration.
- ϵ-$first$ policy. search. The agent will select random actions during exploration for x episodes and will then switch to exploitation.
- Contextual ϵ-*greedy* policy. The context of in which the agent is located with respect to the environment determines whether the agent will employ and ϵ-*greedy* or a *greedy* approach. If a critical situation occurs the agent will not explore but rely on what it has already learned to resolve the critical situation.

The ϵ - *greedy* policy is used in most application due to its simple implementation and easier tuning of its parameters.

For reinforcement learning problems with a continuous action space, forms noise is added to the action output to explore. For

3-1-9 Actor-critic methods

Actor-Critic methods are build up of two components as first proposed by Sutton, Mcallester, et al. (1999). As the name suggests the method is now separated into two components: the actor and the critic. The schematic is shown in Figure 3-3. First, there is the actor which uses a parametrized policy to select an action. Second, there is the critic, which evaluates the action to provide a feedback signal to the actor. The motivation behind this construction is to reduce the high variance of the policy gradients by using an actor to represent the policy independently of the value function (critic). actor-critic is thus a combination of the policy gradient methods and value based methods. The actor itself has no access to the reward signal, but only to the state observation. To put this in perspective relative to earlier method the following comparison can be made. Q-learning can be categorized as a critic-only method, because it directly estimates the value of a given state-action pair without a parametrized policy as usually a greedy approach is taken to select the action. Policy gradient methods can be categorized as actor-only method. There is a policy that is directly updated from the reward signal. In actor-critic the critic estimates the value function (like in Q-learning) but instead uses it to update the policy in the direction of most likely improvement, which is the gradient of the value function. The policy is a probability distribution over the actions.

The critic estimates the error using the temporal difference error in Equation 3-15, with V the value function of the critic. So after an action is taken by the actor, the critic evaluates the value function with the new state to determine if the expected result is achieved. A positive error means that selection of a_t in s_t should be encouraged. The error signal produced is used to update the actor policy according to update rule in Equation 3-16. Numerous methods modify the update rule to accommodate different behaviours. The critic is updated according to the selected value-based methods like eg. SARSA.

$$\delta_t = r_{t+1} + \gamma(V(s_{t+1}) - V(s_t) \tag{3-15}$$

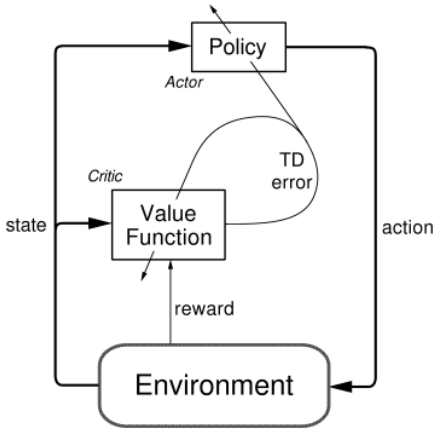


Figure 3-3: Schematic of Actor-Critic Architecture

$$p(s_t, a_t) = p(s_t, a_t) + \beta\delta_t \tag{3-16}$$

The critic in actor critic methods is used to reduce the variance found in policy gradient methods. However, this does come at the cost of some bias due to the approximation of the policy gradient with the use of a value function. A biased policy gradient can fail to converge to the correct solution. To avoid this a value function approximation must be chosen compatible to the policy. Therefore the compatible function approximation theorem by Sutton, Mcallester, et al. (1999) prescribes that a compatible value function approximator must satisfy Equation 3-17 and minimize the mean-squared error to the function parameters w.

$$\nabla_w Q_w(s, a) = \nabla_\theta \log \pi_\theta(s, a) \tag{3-17}$$

To further reduce variance a baseline can be substracted from the critic values eg. by using an advantage function. The advantage represents the relative value of the possible action in a given state (the advantage one action has over another action). This is also useful to prevent discouraging an action that was the best choice when only poor actions are available. The advantage function can be defined as the difference between the Q-value and state value V, because the Q-value of a state-action pair represents the expected discounted reward for that state-action pair where the value function represents the value of the a state. $A(s, a) = Q(s, a) - V(s)$. What is left is the advantage for individual actions with respect to each other. In actor-critic the Q-function is not estimated so it may be approximated using the discounted returns: $A = G_t - V(s)$. A more advanced algorithm to estimate the advantage with lower bias is Generalized Advantage Estimate (GAE) Schulman, Moritz, Levine, Jordan, and Abbeel (2016) following the notation from the paper in Equations 3-18, 3-19 with tuning parameter λ.

$$\delta_{t+k} = \sum_{l=0}^{k-1} \gamma^l * r_{t+l} + \gamma^k * V(s_{t+k}) - V(s_t) \tag{3-18}$$

$$A_t^{GAE} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{3-19}$$

3-1-10 Hierarchical reinforcement learning

Hierarchical reinforcement learning can be described as setting up different abstractions levels of control in a reinforcement learning problem. When using a humanoid navigation example, the lowest level of abstraction could be a controller that moves the joints of the humanoid. A higher level controller can eg. walk to a waypoint by switching between lower level controllers. A higher level controller can thus choose between eg. macros, which is a predefined sequence of actions, or completely different policies that solve a problem. Many flavours exist. A motivation for hierarchical reinforcement learning is to divide a complex problem into simpler sub-problems which helps addressing the curse of dimensionality Durugkar, Rosenbaum, Dernbach, and Mahadevan (2016). Another advantage of having hierarchies is the possibility of longer time horizon strategies to emerge, because the controllers at the higher abstraction level can operate on different time scales than the lower level controllers.

3-1-11 Learning aids

It can be hard for an agent to learn complex tasks from scratch in a complex environment. Therefore the agent can train on simplified environments first and so gradually increase its skill. This is called curriculum learning Narvekar (2017). In a multi-agent competitive setting this is less of an issue as the difficulty of the environment is determined by the skill of the opponent. If agents are co-learning, the difficulty of the environment will gradually increase. In cooperative tasks where full-cooperation is required this is not the case as the task at hand is equally difficult from start to end. In these settings designing a curriculum of task difficulty can speed up learning.

Another way to accelerate learning is reward shaping a.D. Laud and Laud (2004). Providing the correct reward signal to the model while learning can significantly enhance learning. This is also justified from a psychological point of view. With the right feedback you can learn faster.

3-2 Deep reinforcement learning

So far the presented methods are all described as discrete in both state and action space. In order to have a continuous model a function approximator should be utilized. Recent work in reinforcement learning uses deep neural networks as a function approximators for Value functions, Q-functions and parametrized policies. New challenges arise when applying neural networks to reinforcement learning, because the training of neural networks is a supervised learning problem that must be integrated in a reinforcement learning framework.

3-2-1 Deep Q-networks

The search for an end-to-end learning solution in reinforcement learning had a major breakthrough when neural networks were first successfully used to approximate the Q-function, referred to as a Deep Q-Network (DQN) Mnih et al. (2015). Inspiration was taken from image processing techniques where neural networks are used to interpret images for classification and detection. A DQN is used to map the pixel states of video games to actions, which means it uses end-to-end learning. It learns a state representation directly from the raw pixel input and map this state representation directly to state action-values. The DQN architecture is as follows. First convolutional layers are used to reduce the dimensionality of the image data followed by fully connected layers. The output layer has an output number matching the size of the action space. The fully connected layers make use of the rectified linear unit (ReLu) as activation function. The output layer uses linear activation function and outputs the estimated Q-value for the state-action pair.

Approximating the Q-function using non-linear function approximators is unstable as shown by Melo, Meyn, and Ribeiro (1997). Neural networks can therefore not readily replace the Q-table to represent the Q-function. The main reasons for the training instability of the neural network are the correlations present in the sequence of training data when using online Q-learning. The sequential observations are from the same episode and therefore introduce high variance (overfitting) in the function approximation. Typically when training neural networks, a large and diverse dataset reduces the variance of the network output and thus increases its power to generalize over the dataset. In reinforcement learning the distribution of samples is dominated by the action that maximizes the reward and biases the training to the dominant action. The parameters are likely to end up in a non-optimal local minimum or training may even diverge. To tackle this problem Mnih et al. (2015) introduce experience replay to train the network. Each experience is saved as a transition tuple $e_t = (s, a, r, s')$ in the replay memory $D_t = \{e_1, ..., e_n\}$. When training the network a batch of experiences is randomly sampled from the replay memory. This decorrelates the experiences when training the network.

Training a neural network is a supervised learning problem, so the network should have a target value to compute the loss. Therefore off-policy learning is applied with the use of a target network $\hat{Q}$. Off-policy learning is when a separate target policy is used to update the actual policy. $\hat{Q}$ is used to generate target y_j in Equation 3-20 for training the network Q and has the same architecture as the Q-network. Then after the back-propagation step the target network parameters θ^- are updated with the network parameters θ from the Q-network.

$$y_j = r_j + \gamma \max_{a'} \hat{Q}\left(s_{t+1}, a'; \theta^-\right) \tag{3-20}$$

To stabilize training the target network is only updated every c amount of iterations as reinforcement learning requires a stationary environment to learn. This is useful because training a neural network with gradient descent takes a number of iterations to converge. By keeping the target network constant for a longer period of time does stabilize training.

The optimization algorithm is thus very similar to that of Q-learning with the addition of experience replay and delayed updates of the target network. The DQN architecture managed to surpass human level performance in many Atari games.

3-2-2 Double deep Q-networks

DQN is known to be bias to overestimating the Q-values as shown by van Hasselt, Guez, and Silver (2016). The overestimation also occurs with other function approximators even if the method is unbiased. This may no t be a large problem if it still learns the optimal policy. However, as demonstrated by van Hasselt et al. (2016) it slows down learning and in some cases may even hinder learning. The overoptimism is caused by estimation errors. These errors can be introduced by stochasticity of the environment and inaccuracies that are caused because the true values for the function approximations are not known initially. This is mainly due to the fact that only the highest target Q-values are used to update the network. Double Q-learning separates the action selection from the action evaluation by learning two separate value functions, which are updated with separate random experiences. So first an action is selected using a greedy policy. Next this action is evaluated by the second value function to produce the target value for training as also shown in Equation 3-21. It is shown that this in fact reduces the overestimation bias.

$$y_t^{\text{DoubleDQN}} = r' + \gamma Q\left(s', \arg\max(Q(s'; \theta), \theta^-\right) \tag{3-21}$$

3-2-3 Dueling deep Q-networks

Another shortcoming of the DQN is that sometimes the difference in relative value of the Q-values is very small compared to the absolute value of the Q-values. This was one of the reasons to use an advantage function in the actor critic methods. The Dueling DQN architecture implements the advantage function inside the network by using two separate streams for the value function and the advantage function Wang, de Freitas, and Lanctot (2016). When reviewing the definition of the advantage function $A(s,a) = Q(s,a) - V(s,a)$ it can be rewritten to a Q-function, which now also depends on the network parameters θ. The state-value is a scalar value where the advantage and Q-value are vectors with size $|\mathcal{A}|$. Naive implementation of this equation in the network leads to poor performance due to an identification problem. The network will output a Q-value, so when back propagation is applied from the target Q-value it is not identifiable what the value of V and A were, hence there is no use for having an architecture with a separate value and advantage stream. To tackle this problem the advantage is forced to zero for the chosen action by subtracting the maximum advantage value. Alternatively to using the max operator is subtracting the mean from the advantage. These operations are allowed because they do not change the relative rank of the advantage function. In short, this means that the result of the estimated Q-values still yields the same action when applying a greedy policy. The neural network architecture is shown schematically in Figure 3-4

$$Q(s,a;\theta) = V(s;\theta) + A(s,a;\theta) \tag{3-22}$$

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \max_{a' \in |\mathcal{A}|} A(s,a';\theta) \right) \tag{3-23}$$

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s,a';\theta) \right) \tag{3-24}$$

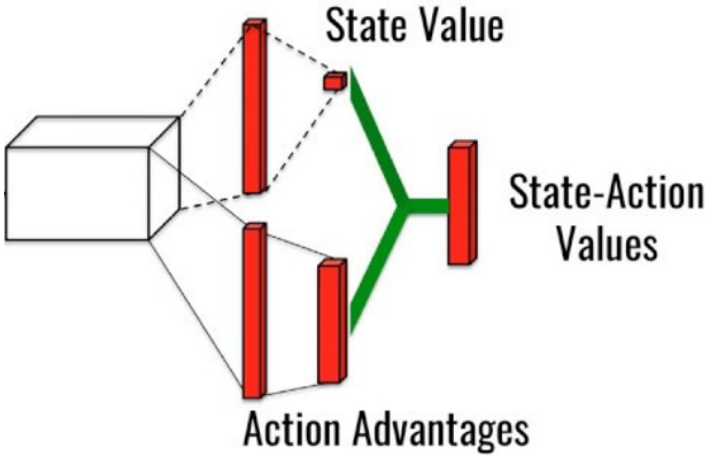


Figure 3-4: Dueling DQN architecture

3-2-4 Deep deterministic policy gradients

Deep Deterministic Policy Gradient (DDPG) Lillicrap et al. (2015) is an adaptation DPG with neural network representation of the policy and critic. To apply the neural network for function approximation, concepts such as experience replay are borrowed from the DQN. DDPG is thus off-policy algorithm. The advantage of the algorithm being off-policy her is the fact that exploration

of the environment is separated from learning. A stochastic policy is used to sample trajectories from the environment (exploration). Think of this as simply adding noise to the output of the policy. These trajectories are then used to train the deterministic policy (learning). Like in DQN a replay buffer is used to store all the transitions. This buffer can be very large because the learning is off-policy. Exploration of continuous actions can be ensured by adding noise to the output of the actor policy.

The DPG and DDPG benefit from a different exploration method than than stochastic policy gradient. Where stochastic policy gradients use a Gaussian noise for action exploration, this is not suitable for the deterministic policy gradient. Take the locomotion environment Brockman et al. (2016) as an example. In this environment humanoids must learn to walk/run a course. The action of an agent is to apply a (muscle) force, which causes an acceleration in the joints. In order to translate this into a change into a position change the acceleration will have to be integrated twice. Since an integrator works as a low pass filter, exploration noise sampled from a Gaussian with zero mean will not make the agent to explore new positions. This is because the noise is filtered by the double integrator. Therefore to ensure proper exploration, noise generated by an Ornstein-Uhlenbeck (OU) process is preferred over Gaussian noise.

An OU is governed by the stochastic differential equation in Equation 3-25. Therefore in an OU process, each timestep is correlated to the previous timestep and is not merely a sample from the probability distribution. The OU process is build up of two components. The first part is pushing the state back to the mean μ, scaled by the parameter β. The second part is a sample from a normal distribution scaled by the parameter σ. Therefore this process will still oscillate around the mean μ. However, due to the correlation of the timesteps an OU process will have a more constant output almost resembling a very noisy cosine of which the 'frequency' that can be tuned using the parameters β and σ. Although there is no real frequency, this can be interpreted as the urgency of returning to the mean value. By tuning the process to the problem OU noise will not be filtered out in the integrators and therefore aids exploration.

$$dX_t = -\beta(X_t - \mu)\,dt + \sigma\,dW_t \tag{3-25}$$

3-2-5 Asynchronous advantage actor-critic

Asynchronous Advantage Actor-Critic (A3C) Mnih et al. (2016) provides an alternative way to decorrelate the training data without using experience replay. The method distributes the learning over a number of independent agents interacting with independently running environments. This diversifies the data samples and allows training on CPU's with a single core for each instance. The distributed system is then kept to a single machine, removing communication overhead found in systems distributed over multiple computers. Memory replay is limited to off-policy methods only, so by applying A3C it is now also possible to apply on policy methods to Deep Reinforcement Learning. Incorporating experience replay on top of A3C could improve the data efficiency, which reduces training times when interacting with the environment is expensive.

3-3 Multi-agent reinforcement learning without communication

In order for multiple aircraft to exist in the TMA the single-agent reinforcement learning techniques have to be extended to Multi-Agent Reinforcement Learning (MARL). Every aircraft can be considered an independent agent. In a multi-agent setting the desired behaviour may be fundamentally different.

3-3-1 Learning type settings

In a multi-agent setting agents can interact with certain objectives. The setting can be cooperative, competitive or a mixed. The specifics of these settings are addressed here.

The air traffic management problem in that aspect is a environment where multiple-agents (aircraft) interact while attempting to reach their destination in a fly to their destination in an efficient. The operation is not fully cooperative. In a fully cooperative setting, all agents will receive the same global reward rather than the local reward earned by an individual agent. This global reward can for instance be the worst reward of any individual agent. This forces cooperation to maximize all rewards. In a fully competitive setting the agents will receive only the local reward and try to optimize their local reward. The Air Traffic Management (ATM) task can be described as a fully cooperative setting. Even though individual pilots may not desire an optimal global reward, from the perspective of an air traffic controller this is desirable.

3-3-2 Challenges in multi-agent reinforcement learning

A number of challenges arise when moving to the multi-agent domain. First of all there is variable amount of agents in the simulation during runtime as aircraft enter the TMA and land. This means the reinforcement learning architecture should be able to handle this. Second, is the credit assignment problem. Global rewards give no information about the utility of individual agents' actions. Third there is the awareness of other agents' information states. Furthermore, MARL implementations suffer from the curse of dimensionality. That is the discrete state-action space grows exponentially with the number of agents as it represents all possible state-action. This means that the computational complexity grows exponentially with the number of agents, making learning difficult and prohibitively slow.

To make the step from single-agent reinforcement learning to multi-agent reinforcement learning (MARL) some additional issues arise. The most straight-forward approach to implement MARL would be to put multiple agents in an environment and have them learn independently as if they were single-agents using Q-learning or DQN. This does not work in practice, because it violates the Markov Assumption of a stationary environment. The non-stationary violation comes from the fact that each agent tries to learn a policy based on its own observations from the environment. Thus, each agent updates its policy based on the actions of concurrent agents. Due to the changing of the behaviour of other agents the environment is non-stationary from the perspective of an individual agent. Therefore the memory replay required for stabilizing DQN learning cannot be applied.

3-3-3 Multi-agent deep deterministic policy gradients

Multi Agent Deep Deterministic Policy Gradients (MADDPG) is an extension of DDPG to the multi-agent setting. It tackles the non-stationary environment issue by implementing centralized training with decentralized execution Lowe et al. (2017). It exploits of having central knowledge of all policies. The schematic of shown in Figure 3-5. Having knowledge of the actions of other agents (by inferring their policy) makes the environment stationary. This means experience replay techniques are applicable again. This does however, assume that the policy of all other agents is known. This is possible at training time by centrally training all policies at training time and executing decentralized at test time. This is achieved by simultaneously training a function $Q(a_1, a_2, ..., a_n)$ which takes as input the actions of all the agents. The action represents the inference of an agents policy, because a policy is just a mapping from state to action. By using this function during training all agents have knowledge on how the other agents are evolving as well. This makes the environment stationary. To make an agent more robust to changes in other agents policy a policy ensemble K of consisting of k sub-policies

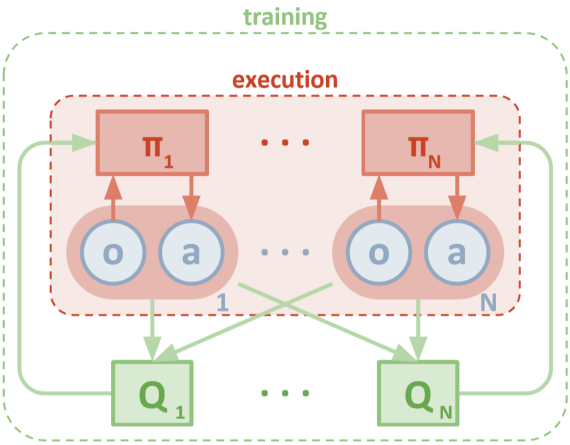


Figure 3-5: MADDPG training vs. execution phase Lowe et al. (2017)

is introduced. These are trained side by side. Every timestep a policy from the ensemble is randomly chosen. At execution time the policies are sampled. For every policy of every agent a replay memory must be kept to update. The disadvantages of MADDPG are that it does not scale well to larger number of agents and a fixed number of agents is required at runtime to perform the centralized training in a stationary environment, because the input of the function Q has a fixed size, so the number of agents present at runtime must also be fixed.

3-3-4 Event-driven multi-agent reinforcement learning

An event-driven multi-agent decision process is proposed by Menda et al. (2018). Rather than learning all the low-level actions it may be more beneficial to learn higher-level actions or macro-actions. This is sometimes also referred to as hierarchical reinforcement learning to classify tasks to multiple levels of abstraction. A most primitive action for aircraft would be to control the actuators of the aircraft to fly it. Then a higher level policy would utilize these primitive policies to perform e.g. a turn. Macro-actions are characterized by the fact that they temporally extended. For example, a left turn with an aircraft is performed by issuing a new heading command. The action is finished once the aircraft is on it's new heading. This raises an issue in a multi-agent setting because agents will have to act asynchronously when executing macro-actions. The Macro Decentralized Partially Observable Markov Decision Process (MacDEC-POMDP) is optimized using an extended Paramater Sharing Trust Region Policy Optimization (PS-TRPO) algorithm while preserving the continuous action and state space without requiring expert-demonstrations. This means that the agents share parameters and must consequently be homogeneous.

An issue that rises when using event-based actions in a multi-agent environment with discretized time is the possibility of a race condition. This is when multiple events occur within a discrete timestep. Temporal information about the events is lost, because it is impossible to know which one occurred first. This could potentially cause issues when the actions are not working together. Since BlueSky is a fixed timestep simulator, care has to be taken to prevent race conditions or handle them well.

3-3-5 Proximal policy optimization

A more advanced family of algorithms that can be classified as a policy gradient method fall under the name Proximal Policy Optimization (PPO) by Schulman, Wolski, Dhariwal, Radford, and Klimov (2017). PPO are trust region methods. This means that policy gradient is computed using a 'surrogate'

objective function. The update size in the maximization process is constrained to a region in which the update size is trusted. This ensures that the new policy does not diverge from the existing policy in the update step and thereby ensuring near monotonic performance improvements. The new policy will be in the 'proximity' of the old policy. Trust Region Policy Optimization (TRPO) uses the Kullback-Leibler (KL) divergence to constrain the policy updates as shown in Equation 3-26. A downside is that this is difficult to implement and requires expensive matrix inversions to compute the KL constraint.

$$\begin{aligned} \text{maximize } L^{CPI}(\theta) &= \hat{\mathbb{E}}\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}\left[(r_t(\theta)\hat{A}_t\right] \\ \text{subject to } &\hat{\mathbb{E}}\left[\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \end{aligned} \tag{3-26}$$

Instead of using a KL divergence constraint, PPO suggests using simpler constraints. Most simple and effective is applying a clipped objective function. This enforces a pessimistic lower bound on the gradient similar to the KL penalty and reduces the gradient computation to a first order approximation within the clipped region with clipping hyperparameter ϵ to define the clipping range.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \tag{3-27}$$

To train the network a trajectory will be sampled by interacting with the environment, after which multiple batches of gradient descend can be run on the policy network. The new policy is then used to sample a new trajectory. A prerequisite is that an estimation of the advantage must be computed to calculate the surrogate loss and subsequently the policy gradients.

3-4 Multi-agent reinforcement learning with communication

Another way to try speed up learning and reduce the curse of dimensionality found in MARL is to use a communication channel between the agents. If agents communicate essential information through a dedicated channel, there is no need for every agent to know all the observation and states of all the other agents to obtain sufficient information about the intended actions of other agents. In the reinforcement learning community new challenging benchmarks are being setup. One of these is StarCraft Vinyals et al. (2017). Starcraft is a game where a player has to control multiple units in an army to kill the enemy army. The environment is characterised by a dynamically changing army size as well as the need for cooperation within the army. In this section first in a short introduction on sequence modelling is given followed by two notable approaches that use inter-agent communication to StarCraft. Another advantage of using a communication channel is that the output shape of the communication channel is independent of the number of agents communicating over the channel. Therefore the number of agents can be changed dynamically at runtime without compatibility issues.

3-4-1 Sequence data modelling

For communication among multiple aircraft in a decentralized approach or modelling a sequence order, neural network may play an important role. In Natural Language Processing (NLP), where temporal dependency of sequences play an important role, a Recurrent Neural Network (RNN) is used to build a language model. A sequence with length T_x is defined by a set of input vectors $\mathbf{x} = \{x^{<1>}, x^{<2>}, ..., x^{<T_x>}\}$ where $<>$ represents the sequence number. The RNN will output a sequence with length T_y

[3]Andrew Ng, 'Coursera Deep Learning Specialization - Sequence Models', [Lecture Notes]

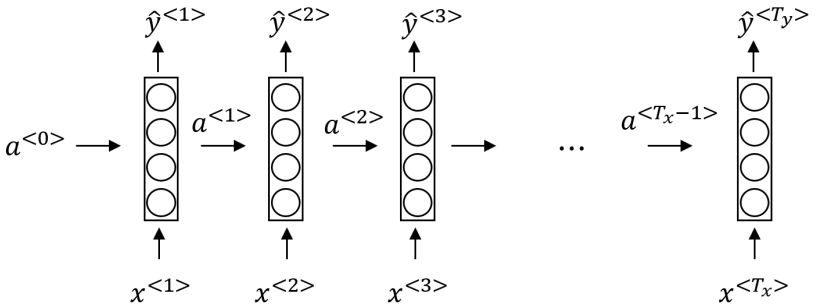


Figure 3-6: Unrolled recurrent neural network [3]

An RNN Pineda (1987) takes two inputs, namely the entry of a sequence $x^{<t>}$ and hidden state $a^{<t-1>}$ at time step t where $a^{<0>}$ is usually initialized with a zero vector. It outputs a new hidden state $a^{<t>}$ and outputs estimate $\hat{y}^{<t>}$. Contrary to standard neural networks, a RNN has multiple input and output vectors. Thus, every input/output pair has its own set of network weights. W_{ax}, W_{aa}, W_{ya}. This means the equations for forward propagation are presented Equations 3-28, 3-29 for network layer i and biases b and activation function g. The activation function for the hidden states are usually tanh or Rectified Linear Unit (ReLU). Common activation functions in the output layer are sigmoid or softmax depending on the intended purpose of the network. As an RNN models temporal dependencies the backpropagation step is called backpropagation through time. The loss function used is the cross-entropy loss in Equation 3-30, which is also used in binary classification tasks.

Equations

$$a^{(i)<t>} = g(W_{aa}^{(i)} a^{(i)<t-1>} + W_{ax}^{(i)} x^{(i)<t-1>} + b_a^{(i)}) \tag{3-28}$$

$$\hat{y}^{(i)<t>} = g(W_{ya}^{(i)} a^{(i)<t>} + b_y^{(i)}) \tag{3-29}$$

$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log(\hat{y}^{<t>}) - (1-^{<t>}) \log(1 - y^{<t>}) \tag{3-30}$$

$$\mathcal{L}(\hat{y} \cdot y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) \tag{3-31}$$

RNNs exist in many forms. There are the many-to-many architecture where every timestep the network also produces an output so that $T_x = T_y$. Many-to-one architectures also exist as well as one-to-many. RNNs only utilize information earlier in the sequence due to information passing in one-direction through the network. Hence, there is upstream information available and a certain hierarchy. When not dealing with time sequences it may also be useful to include information from upstream in the sequence. This is what a Bidirectional Recurrent Neural Network (Bi-RNN) does Schuster and Paliwal (1997). The Bi-RNN is a network composed that simultaneously processes information from the left side of the sequence as well as from the right side of the sequence. The outputs of both traversal directions are combined to produce an output composed of both upstream and downstream information.

3-4-2 BiCNet

The first approach to achieving human-level performance in the learning to play StarCraft combat games is with the use of a Bidirectional Communication Network (BiCNet) Sukhbaatar, Szlam, and Fergus (2016). BiCNet makes use of a Bi-RNN, see Figure 3-7. An actor-critic approach is used where both the actor and critic use a BiCNet architecture. Bidirectional neural networks take sequence data as input. For the actor the sequence in this case is series of embeddings of each agents' observations.

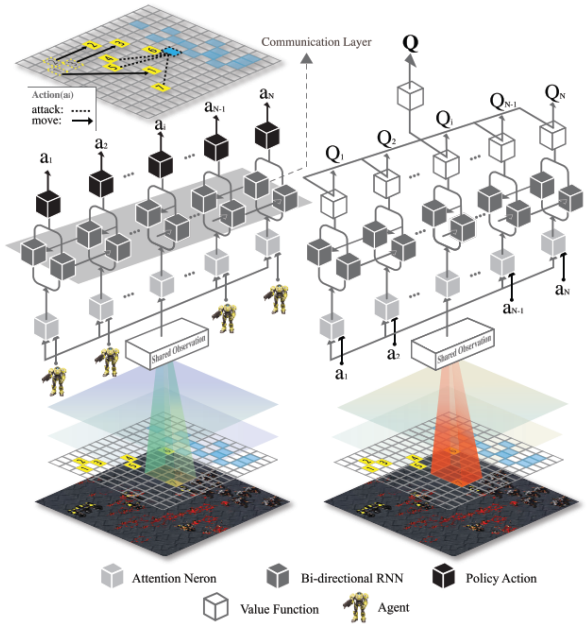


Figure 3-7: BiCNet architecture

By using a many-to-many architecture for the network, every sequence entry produces an output corresponding to the actions. The critic takes as input a shared observation as well as the actions chosen by each agent and outputs a Q-value for every agent as well as a global value used to determine the global value. To train the network a multi-agent deterministic policy gradient approach is taken applied to the actor-critic framework as opposed to the centralised training decentralized execution method found in MADDPG as also mentioned by Sukhbaatar et al. (2016)

3-4-3 CommNet

The Communication Network (CommNet) proposed by Sukhbaatar et al. (2016) uses an explicit communication model independent of the number of agents participating in the communication. CommNet is composed of a series of communication steps. The CommNet architecture is presented in Figure 3-8. For every communication step an agent is represented by a module f^i where i denotes the communication step. The mean of all module outputs is taken as a communication signal. Every module has the communication signal c_j^i and the previous communicaton step module output h_j^i which are multiplied by their respective weights matrics and then fed through a tanh activation function to produce h_j^{i+1}. The CommNet model ϕ is composed of n communication steps $T^i, ..., T^n$. The communication steps can also be replaced by 'timesteps' of RNNs. In order to train the network policy gradient methods are used. The architecture has been succesfull at solving traffic junction tasks with partial observability and Sukhbaatar et al. (2016) also applied it as a baseline for their StarCraft benchmark.

3-4-4 Algorithm Overview

This section provides an overview of the algorithms presented and their applicability to different problem domains. The overview is given in Table 3-3.

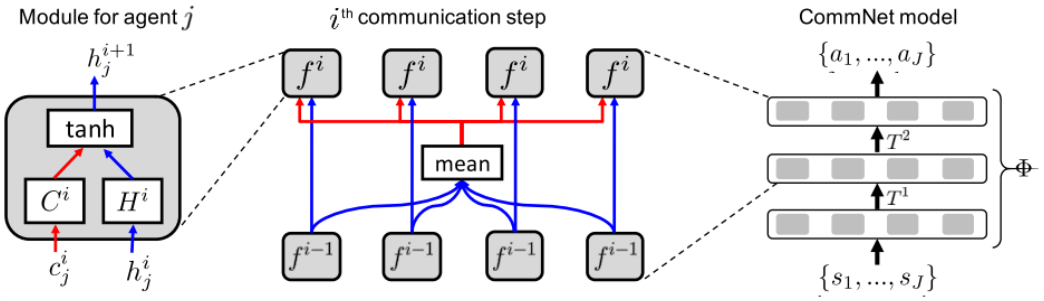


Figure 3-8: CommNet architecture

Table 3-3: Overview and classification of reinforcement learning algorithms. D is for Discrete and C is for Continuous

Algorithm	Type	on/off-policy	State space	Action space	Exploration
Q-learning	Value-based	off-policy	D	D	ϵ-greedy
SARSA	Value-based	on-policy	D	D	ϵ-greedy
DPG	Policy gradient	on-policy	C	C	OU noise
DQN	Value-based	off-policy	D and C	D	ϵ-greedy
Double DQN	Value-based	off-policy	D and C	D	ϵ-greedy
Duelling DQN	Value-based	off-policy	D and C	D	ϵ-greedy
A3C	Actor-critic	on-policy	D and C	D and C	ϵ-greedy and Gaussian noise
DDPG	Actor-critic	off-policy	C	C	OU noise
PPO	Policy gradient	on-policy	D and C	D and C	ϵ-greedy and Gaussian noise
Commnet	Policy gradient	on-policy	D and C	D and C	ϵ-greedy and Gaussian noise

3-5 Automation in TMA for ATC

The terminal manoeuvring area is a complex airspace where traffic streams have to be merged and sequenced into one or two traffic streams in for landing, depending on how many runways are in use. Many approaches have been taken automate this sequencing and spacing process. Although the solution presented in this section do not use machine learning techniques to address the sequencing and spacing, the problem breakdown can be useful. A lot of research has been performed in this area so only a few notable once will be mentioned.

3-5-1 Terminal AutoResolver

The Terminal Auto Resolver (TAR) by Nikoleris, Erzberger, Paielli, and Chu (2014) is an iterative solver for the sequencing and spacing problem. It works by probing the 4D aircraft trajectories for conflicts when a new aircraft is eligible for scheduling. If a conflict is detected a set of resolutions are generated using pool of manoeuvres regularly issues by ATC. Some moves are preferred over others due to their simplicity in execution. They are listed below with an explanation of the effect they have and why they may be used in Table 3-4. Next, the generated trajectories are then probed for conflicts again. The best trajectory is chosen, but if no feasible solution was found an additional iteration is performed. Also runway reassignments can also be a tool if the incurred delay becomes too large. The assumption is that the aircraft 4D trajectory is known and aircraft thus fly a predefined routes to the runway.

Table 3-4: Confict resolution manoeuvres

Speed change	The speed change is usually a speed reduction as there are usually maximum speed restriction enforced in the TMA.
Hold speed	The aircraft will hold its current Calibrated Airspeed (CAS) longer than required by the nominal arrival trajectory. This allows the aircraft to arrive at the Final Approach Fix (FAF) earlier if necessary without violating maximum speed restrictions.
Offset of base leg	The extension of a leg when approaching a downwind turn. Often combined with speed reduction when speed reduction alone is insufficient.
Path stretch	Path stretches are used to lengthen the path of the trailing aircraft to resolve a conflict. Two types of path stretch manoeuvre exist: symmetric path stretches and elliptical path stretches. Elliptical path stretches may be used when a symmetric path stretch does not resolve a conflict while preserving the aircraft's Scheduled Time of Arrival (STA).
Horizontal Vector Turn	This manoeuvre is a resolution manoeuvre intended for short range use that takes into account the turn rate of an aircraft by specifying an auxiliary waypoint that aircraft have to fly over.
Delayed turn back for departures	Much like the offset of base leg manoeuvre this can be used for departures with a large heading change in the departure route.
Fanning	This is an extension of the final approach. The final approach is a straight segment from the final approach fix to the runway. When fanning this segment is extended. The effect is similar to a path stretch.
Compound horizontal manoeuvres	These include combination of above mentioned horizontal manoeuvres.
Temporary altitude climb	Temporarily hold altitude during climb/descend to resolve a vertical conflict.

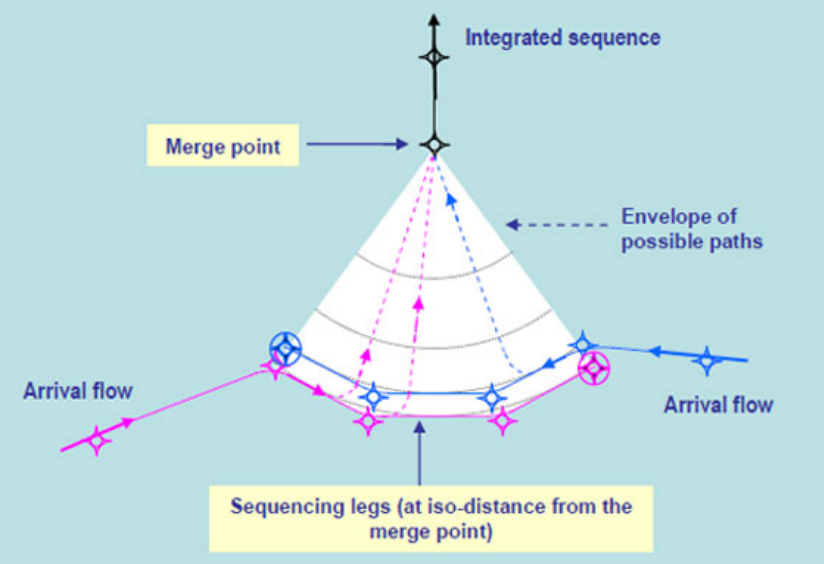


Figure 3-9: Schematic overview of the Point Merge System Man (2015)

3-5-2 Point merge system

In Europe research has been done to a new system to allow continuous decent approaches to be performed. This resulted in the PM system where a single merge point of multiple traffic flow is defined. Arriving aircraft will fly along a semi-circle around the merge point, the sequencing legs, until they are cleared to proceed to the merge point and subsequently the runway. Advantages of this arrival structure are a reduced workload and a more predictable traffic flow. The continuous descend approach allows for fuel saving and noise reduction. [4]. The point merge has already proven its operational benefits as it has been implemented at many airport in Europe.

3-5-3 Sequencing and spacing with genetic algorithms

Another approach to strategically optimize the sequencing and spacing of air traffic is with the use of genetic algorithms. Zuñiga, Delahaye, and Piera (2011) defined main approach routes with alternative routes spaced sufficiently far apart to ensure separation. Using genetic algorithms an optimal route within the constraint route system are found for every aircraft while ensuring optimal runway throughput. The sequencing and spacing problem could be solved effectively around Mallorca, Spain.

3-6 Application of reinforcement learning in ATM

This work is not the first to apply reinforcement learning methods to ATM. Alves, Weigang, and Souza (2008) were among the first to apply reinforcement learning in ATM. A decision support system was built for tactical air traffic flow management. The task of the system is to provide flow balancing resolution over a set of sectors to prevent sector saturation. This is comparable to structuring aircraft in the TMA, but at a higher abstraction level. Meta-level control is used to deliver a flow distribution

[4] F.J.M. Wubben and J.J. Busink, 'Environmental benefits of continuous descent approaches at Schiphol Airport compared with conventional approach procedures', 2000, National Aerospace Laboratory NLR

over the sectors. Both egalitarian distribution policies and distribution with flow prioritization were applied successfully.

Attempts to automate air traffic control tasks using machine learning have been made. Crespo, Weigang, and de Barros (2012); Tumer and Agogino (2007) both use a reinforcement learning controllers to delay upstream traffic to manage the congestion of airspace upstream. Tumer and Agogino (2007) uses a multi-agent reinforcement learning controller. The agents are distributed as fixtures rather than aircraft. The main advantages are that this provides a scalable approach. The fixtures control the 'miles in trail' of the aircraft passing over them. In this way the agents can introduce delays into the system. Focussed on the delaying air traffic downstream to manage the congestion of the airspace.

Another application to air traffic flow management is proposed by Tumer and Agogino (2007). To manage the congestion a reinforcement learning controller learns to delay traffic upstream. The agents in this system are located at fixtures. The main advantage is that this provides a scalable approach. A global reward and local 'difference' reward for an agent to learn is specified. The agent action is to set the amount of spacing the aircraft should have when arriving at the node. The aircraft will have to ensure they meet this requirement. The Q-learning algorithm is used to learn an optimal policy. Usin this method the air traffic flow could be successfully balanced to prevent congestion.

Then moving from air traffic flow management to merging of 4D traffic on the point-merge system. An agent-based approach to automatically merge 4D trajectories is presented by Man (2015). The aircraft agents, a 4D trajectory planning agent, a flow manager agent and a Conflict Detection & Resolution (CD&R) agent work together to merge the air traffic flow into a landing sequence. The air traffic flow manager is modelled as a reinforcement learning agent situated on the Merge point. The agent learns the required times in trail for multiple aircraft, much like the situation much like the approach taken Tumer and Agogino (2007) which used miles in trail for en-route traffic separation on merge points.

Finally, an approach towards the completely automated sequencing and spacing using reinforcement learning techniques using deep reinforcement learning is presented by Brittain and Wei (2018). Hierarchical reinforcement learning is applied to a simplified sequencing and spacing problem on a web-based game developed by NASA called Sector33. In the game the player must control the route and speed of aircraft to ensure they arrive at the final approach fix at a specified time. In order to do this a nested controller is used. The high-level controller selects the route at the start of the level. The lower-level controller then sets the aircraft speed at fixed time intervals. Thus, both controllers operate at different time scales. The episode terminates early when aircraft collide. The algorithm could find optimal solutions to beat the game. The controllers are trained using the DQN algorithm. Although this it is an achievement still important limitations apply. The controller does not generalize to other situations as it learns to control a fixed number of aircraft in a relatively simple environment. These are obstacles that must be overcome.

All the approaches mentioned provide ways to automate the sequencing and spacing of aircraft in the TMA or have contributed towards this goal, but none of them are engineered specifically with the goal to study emerging strategies learned by the controller. When reviewing the latest developments in the reinforcement learning community there is a move towards general intelligence, which is also the focus of this research. The application of these techniques to air traffic control could be the next step in automation of air traffic control tasks.

Chapter 4

Preliminary Analysis

Preliminary analysis has been performed to help answer the first research subquestion 'What reinforcement learning techniques are suitable to learn the sequencing and spacing tasks in the terminal area?'. The analysis is performed using the BlueSky Open Air Traffic simulator by Hoekstra and Ellerbroek (2016). The goal of the preliminary analysis is to establish an understanding of the nuts and bolts of reinforcement learning as well as to explore the possibilities for a suitable representation of the sequencing and spacing of aircraft in a reinforcement learning framework and to try out some of the methods described in chapter 3. Methods applied are the Multi-Agent Deep Deterministic Policy Gradients , Deep Q-Networks and the Dueling Deep Q-networks, Deep Deterministic Policy Gradients while also exploring network representations as well as a suitable problem representation.

4-1 Conflict resolution using MADDPG

4-1-1 Experiment description

A very first experiment was performed to see how to implement a reinforcement learning algorithm in BlueSky. The problem representation was very simple. 3 aircraft are spawned in close proximity in a conflict. The goals was to solve the conflict without violating minimum separation limits. Negative reward is given for violating the minimum separation limits. The episode terminates if the aircraft have not crashed after 100 timesteps. The aircraft states are $\mathcal{S} = \{\text{lat}, \text{lon}, \psi\}$ and the actions were $\mathcal{A} = \{\text{left}, \text{straight}, \text{right}\}$ where the aircraft would perform the action until selecting a new action at the next timestep. The size of the timestep does determines how far the aircraft turns per action selection.

4-1-2 Method

The method used to solve this problem was MADDPG. As a first experiment the MADDPG Lowe et al. (2017) is implemented in BlueSky, which uses a centralized learning and decentralized execution. This means that during learning the aircraft have access to each others policy. In that way they have knowledge on what actions the agents are going to take when choosing their actions. When the algorithm is deployed the agents do not have access to each others policy but will have learned implicit behaviour.

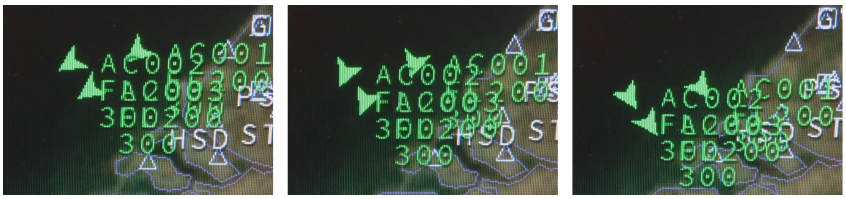


Figure 4-1: Coordinated turning manoeuvre by 3 aircraft observed when training with MADDPG

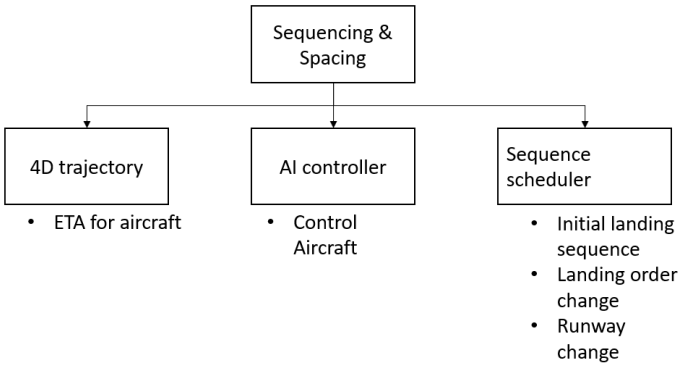


Figure 4-2: Component break down of the problem representation

4-1-3 Results

The aircraft have learned to fly in a coordinated fashion, by al choosing to turn left to solve the problem after a short period of training. In retrospective, however, the exhibition of this behaviour after such a short learning period is very likely caused by divergence of the Q-values to NaN as was sometimes also encountered when using DQN as will be discussed later. The actual behaviour was never fully analysed because another restriction of MADDPG arose that was too large a restriction to continue investigating the algorithm. It was not possible to change the number of flying aircraft during runtime, because the number of aircraft were embedded in the network architecture during the 'centralized learning' phase. The formation flying is shown in Figure 4-1. This marks the first challenge in finding a suitable problem representation combined with a reinforcement learning algorithm: How to handle a variable amount of aircraft at runtime.

4-2 4D trajectory navigation with DQN and Dueling DQN

In search for a suitable problem representation the following problem setup has been investigated. When reviewing what components existing automation systems for sequencing and spacing are composed of a set of useful components has been identified. In Figure 4-2 the components of potential sequencing and spacing automation are displayed. The first component is 4D trajectory predictor. Its task is to predict the time of arrival for aircraft at keypoints along the route. This could be a waypoint eg. the FAF. The second component is the AI controller, which will control the aircraft by choosing action according to a policy learned with reinforcement learning. Finally there is the sequence scheduler or Arrival Manager (AMAN). Its task is to determine the landing sequence and assign a scheduled time of arrival to aircraft and thereby satisfying the minimum wake separation requirements. This AMAN can be a controller following a deterministic algorithm for generating the sequence or also be a controller trained using a machine learning algorithm.

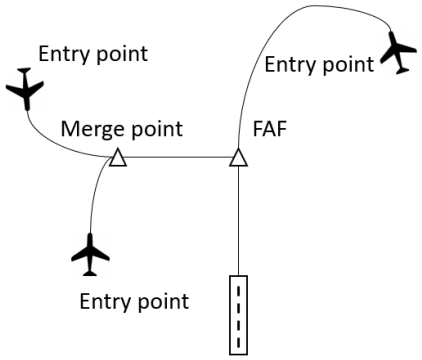


Figure 4-3: Airspace layout for problem design

With these components of the system established the details of how the reinforcement learning is going to control the aircraft is still undefined. The following airspace structure is proposed with the aid of Figure 4-3. Aircraft can enter the experiment area at the boundary of the experiment area. From the entry point the aircraft must fly towards the FAF or any other merge point that is defined along the route towards the FAF. The sequence scheduler will assign the aircraft a STA at which the aircraft must arrive at this waypoint. The aircraft must than make sure the aircraft arrives at this waypoint at the designated time while maintaining the minimum separation criteria with the aircraft around it. In this setup the responsibility of maintaining sequence spacing and the responsibility to adhere to minimum separation criteria is divided over the the sequence scheduler and the reinforcement learning controller respectively. Without further specifying how the sequence scheduler will work, the current description of the reinforcement learning controller is experimented with.

4-3 Experiment description

The goal of the experiment is for a reinforcement learning controller to learn to navigate an aircraft towards a goal while incorporating time restrictions. The aircraft enters the simulation at the Initial Approach Fix (IAF) in the EHAM TMA to navigate to the final approach fix to make an approach at RWY06 of Schiphol airport. When arriving at IAF SUGOL the aircraft is assigned a STA as to mimic an arrival manager. The task of the agent is to arrive at the FAF at the STA. So the agent must simultaneously learn to navigate to the FAF at waypoint EH609 and to absorb a delay assigned at the start of the episode. The delay is t_{delay} is sampled randomly from a uniform distribution from the range [0, 100] seconds. So sometimes the aircraft is ordered to fly a direct route towards the waypoint, while at other times it must find a way to delay its arrival.

The agents observation is a set of augmented states. $\mathcal{S} = [d, t_{delay}, \psi_{rel}]$ with d the distance to the target waypoint, t_{delay} the time to still be absorbed and ψ_{rel} the relative heading to the target waypoint as shown schematically in Figure 4-4. ψ_{rel} is normalized to the range [-1, 1] where 0 means flying straight towards the target. The agent does not need to know its actual position with respect to the target waypoint, it only needs to know what it's position is relative to the waypoint in order to have an incentive of what direction to fly in. This does mean that the state of the aircraft can be ambiguous with regards to absolute position. The absolute position is however, not relevant to the experiment. t_{delay} is computed by making use of 4D trajectory prediction. A deterministic 4D trajectory predictor plug-in is written for BlueSky. The 4D trajectory predictor computes the time to a waypoint by making use of the flight plan of the aircraft. All altitude constraints and speed constraints are recorded in the flight plan. Then the arrival times at every waypoint is computed based on the behaviour of the autopilot for both lateral and vertical navigation and the aircraft

Table 4-1: Reward function components

State	Description	Gradient	Bias
d	Distance to FAF	α_d	b_d
t_{delay}	Time to absorb before arrival	α_t	b_t

performance and flight phase in BlueSky. If the aircraft heading is not aligned with the next waypoint in the flight plan, the 4D trajectory predictor will first compute the shortest turn to the desired heading iteratively and correct for the additional flight time. The 4D trajectory predictor allows the state t_{delay} to be computed at each timestep during an episode by computing the shortest flight time from the current position to the FAF within the constraints of the flight plan.

Next the action for the agent. To keep the experiment simple the action space of the aircraft is limited to heading changes and flying straight. $\mathcal{A} = \{\text{left}, \text{straight}, \text{right}\}$ Inspiration in the action selection is taken from robotics where robots tasked with navigation in unknown environment Sharma and Taylor (2012). Robots have to choose a waypoint on a grid inside their field of view to navigate to. The lower-level path planning navigates the robot to the waypoint. The aircraft will also choose a waypoint to navigate to. It has three options: Fly straight for 0.25nm change heading by 15 degrees left or change heading by 15 degrees to the right. A path planning module will then compute the location of the waypoint that coincides with the termination location of the action. When the aircraft reaches the waypoint a new action has to be chosen. This approach fits well into the design of BlueSky. When using heading changes there will be no active waypoint in BlueSky, which restricts the use of the VNAV module for vertical navigation. With the perspective of BlueSky handling vertical navigation in more complicated scenarios it is desirable to add new waypoints to the route. The autopilot in BlueSky will then navigate the aircraft to the calculated waypoint. A schematic of the action space is shown in Figure 4-5. These actions are temporally extended because they do not follow a fixed time step. Therefore the problem is modelled as a SMDP for this particular setup.

$$r_t = \alpha_d d + b_d + \alpha_t t + b_t \tag{4-1}$$

Finally a reward function must be specified to supply the action feedback to the model. As the reward function has been altered many times through the experiment runs a basic reward structure can be identified to which modification are made in Equation 4-1. The reward is constructed of three components which are the states also used in the simulation and are shown in Table 4-1. Distance is important because the aircraft should learn to fly towards the FAF. By providing a reward based on the distance from the FAF the aircraft is given an incentive to fly towards the FAF. Time is important because the main goal of the experiment is to arrive on time. Finally heading is important to have the aircraft arrive at the correct heading with respect to the runway. Rewards can either be sparse or dense, eg. only given when agent reaches the FAF or continuously each timestep. The reward should be well balanced to prevent reward hacking by the agent. Reward hacking occurs when the agent exploits flaws in the reward signal that lead to undesired results. The episode terminates when the aircraft has reached the FAF or when the aircraft is no longer on track to meet the experiment objective to speed up learning. Exploring the environment will be done using an ϵ-greedy strategy.

4-4 Method

To solve this environment DQN and Dueling DQN are used. The training algorithm is the same for DQN and Dueling-DQN, but with a different network architectures. the DQN algorithm is used as shown in Algorithm 1. Additional parameters that need tuning in order for the process are called

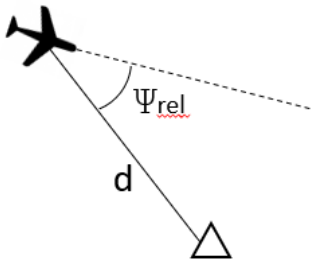


Figure 4-4: State representation

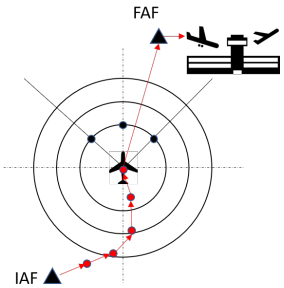


Figure 4-5: Action Space

hyperparameters. Any optimization problem has its own set of hyperparameters to tune in order to obtain satisfactory results. These influence process characteristics as convergence speed as well quality of the steady state error in the optimization process. Reinforcement learning has many hyperparameters and when deep neural networks are applied to reinforcement learning, many more hyperparameters are added to the list. An overview of additional hyperparameters that control the learning process is given in Table 4-2. Additionally, choices for the neural network architecture, loss function and optimization algorithm should be made.

Algorithm 1: Deep Q-networks with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
for *episode=1, M* **do**
  Initialize sequence $s_1 = \{x_1\}$
  for *t=1,T* **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(s_t, a; \theta)$
    Execute action a_t in simulator and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random minibatch of transitions (s_t, a_t, r_t, s_{t+1}) from D
    if *episode terminates at step j+1* **then**
      Set $y_j = r_j$
    else
      Set $y_j = r_j + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'; \theta^{-})$
    end
    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every c steps set target network parameters $\theta^{-} \leftarrow \theta$
  end
end

4-4-1 Results for DQN

The first experiment is performed with DQN. Many runs have been performed to find a set of hyperparameters that were appropriate to find a policy to solve the problem. The DQN were trained in the BlueSky environment. The network weights were saved every 10 episodes. When testing the results

Table 4-2: Hyperparameter overview

Hyperparameter	Description	Origin
Memory size	Maximum number of entries for the replay memory.	Deep reinforcement learning
γ	Discount factor for future rewards. This hyperparameter controls the time horizon in which the agent considers future rewards to be of importance during learning	Reinforcement learning
ϵ	Probability of choosing a random actions	Exploration vs. exploitation
ϵ_{min}	Minimum probability of choosing a random actions during learning	Exploration vs. exploitation
ϵ_{decay}	Annealing factor for the ϵ	Exploration vs. exploitation
λ	Learning rate	Optimization
Batch size	Number of experiences sampled from the replay memory for each training step	Neural network optimization
c	Target network reset factor. Every c training steps, the target network weights are set to the current policy weights to stabilize training.	Deep reinforcement learning.

the saved network weights are loaded into the DQN and the simulation is runs with deterministic action selection for 25 episodes. The best results obtained with DQN are shown in the Figures 4-6 to 4-11. Hyperparameters used along with network architecture can be found in the Appendix A. As can be seen the aircraft learned to consistently fly towards the FAF. In Figure 4-7 the t_{delay} for the aircraft is plotted over time for all 25 episodes. Even the aircraft that have to fly a direct route still loose time. This can partly be considered an artifact the discrete action space. The aircraft can simply not fly in a perfect straight line to the FAF. The aircraft have a tendency to make a manoeuvre early in the approach to absorb the delay. The reason for doing this becomes evident when looking at the reward received at every timestep in Figure 4-8. Higher rewards are given for a lower t_{delay} and d. An early manoeuvre is the fastest way of obtaining a high reward. Then as the aircraft approach FAF the reward continuous to increase due to the d component. Sometimes some time is lost due to overshoot the the STA so t_{delay} becomes negative, for which penalties are given. It can be argued whether this is the smartest approach in a real-life situation, but it is evident that this behaviour is induced by the design of the reward function. The end result is that the aircraft overshoots the desired time of arrival.

Figure 4-11 shows the running average of the final reward received at termination for every episode of training. The closer t_{delay} is to 0, the higher the reward at termination. The best result is obtained after about 1000 episodes of training. This graph can be interpreted as a learning curve for the aircraft. The agent shows some clear signs of converging at least to a local optimum around 1000, 3200, 7500 episodes where a increase in reward over episodes is very clear. In between the policy fails to find a feasible solution to the problem. After 9000 episodes of training performance drops dramatically. During training the Q-values were constantly growing in magnitude, while their relative value remained very close to each other. After 9000 episodes the Q-values diverged to NaN, explaining the sudden drop in performance.

4-4-2 Results for Duelling DQN

In order to suppress the overestimation of the Q-values that may be a possible cause of solution divergence the Dueling DQN algorithm is implemented. The dueling DQN uses advantage estimation inside the network, which should limit the overestimation of the Q-values and also speed up training.

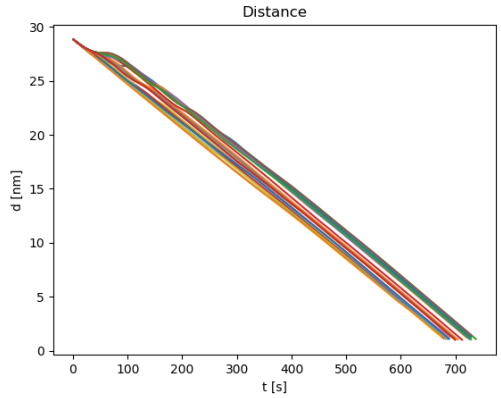


Figure 4-6: Distance vs. time for 25 test episodes after 1000 episodes of training with DQN

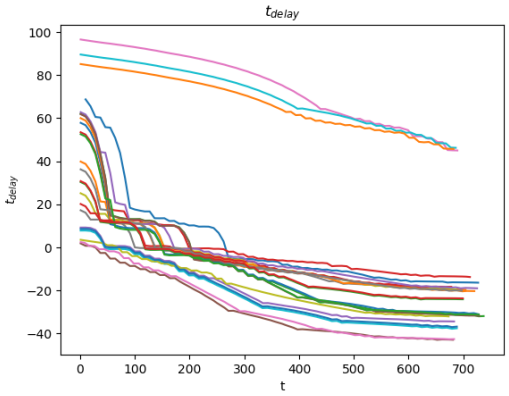


Figure 4-7: t_{delay} vs time for 25 test episodes after 1000 episodes of training with DQN

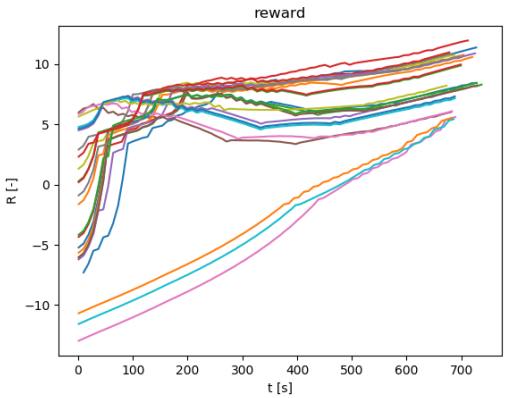


Figure 4-8: Reward vs. time for 25 test episodes after 1000 episodes of training with DQN

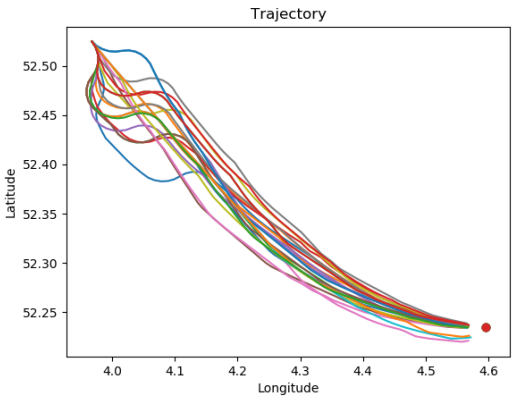


Figure 4-9: Trajectories for 25 test episodes after 1000 episodes of training with DQN

Again hyperparameters en setup for this run can be found in the Appendix B. A different set of hyperparameters and reward function design is used in the Dueling DQN setup as in the DQN setup due to improvements on earlier iterations. Therefore it would not be fair to compare the two methods in this experiment. For comparison of the performance of these two methods please refer to Wang et al. (2016). Again the best results obtained with the Dueling DQN method are presented in Figures 4-12 to 4-16. The reward function in this setup has been changed to give high penalties when the aircraft arrives too late at the FAF. It is interesting to see how this has affected the behaviour of the aircraft. When looking at the trajectories flown by the aircraft in Figure 4-15 it is evident that these show many similarities to path stretch techniques and looks a lot like a real approach from SUGOL to EHAM RWY06. This also shows that the aircraft has become more conservative in the way it absorbs the delay. No aggressive early manoeuvre but gradual absorption through path stretch. Still, the aircraft has a bias to overshoot the STA. The penalties are clearly reflected by the sudden drops in the rewards received over time in Figure 4-14. The resulting performance is more consistent as can be seen by the compactness in box plot in Figure 4-16. For two runs the reward and t_{delay} suddenly drop for the final timestep. To calculate t_{delay} also a turn is iteratively computed towards the next waypoint. The calculation did not converge if the aircraft is too close to the waypoint for a normal

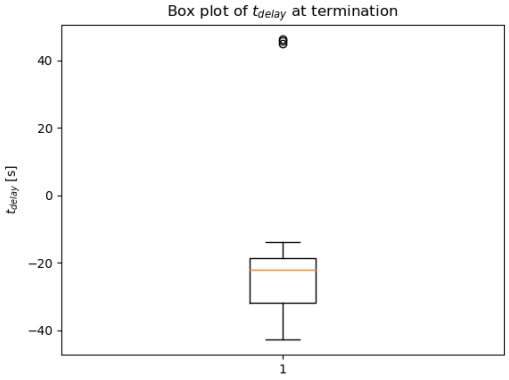


Figure 4-10: t_{delay} at termination for 25 test episodes after 1000 episodes of training with DQN

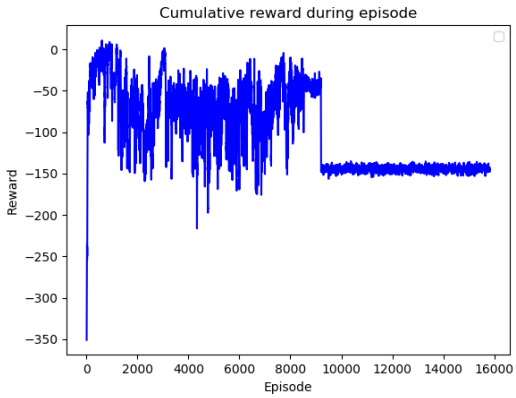


Figure 4-11: Running average of 10 samples for the final reward received during an episode while training with DQN

turn to be calculated. This is undesired as it adds noise to the reward signal and should be addressed.

Figure 4-17 shows the learning curve for the Dueling DQN architecture. The graphs do not show the same metric as Figure 4-11, but shows the running average of cumulative reward collected during an episode during training. The interpretation is similar. The Dueling DQN learns faster than the DQN, which was observed consistently between runs. The highest performance is observed at 375 episodes of training. Although the overestimation of the Q-values in Dueling DQN setup was less the training was still unstable as can be deduced from the large fluctuations in the learning curve.

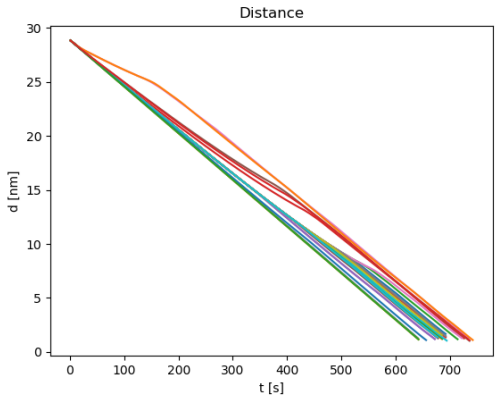


Figure 4-12: Distance vs. time for 25 test episodes after 375 episodes of training with dueling DDQN

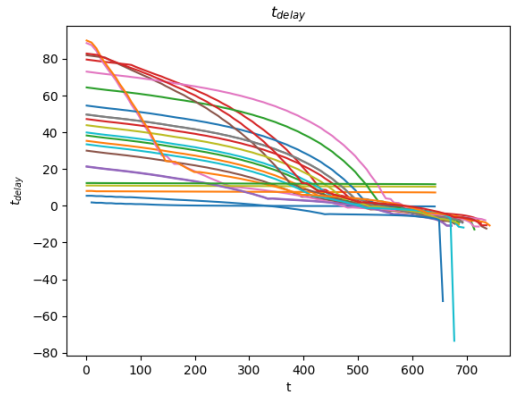


Figure 4-13: t_{delay} vs time for 25 test episodes after 375 episodes of training with dueling DQN

4-4-3 Discussion

Even though the results presented look very promising, obtaining these result was far from trivial. As mentioned before the reward function should be well balanced to prevent reward hacking. During training some reward hacking examples have been observed. To speed up learning, it was thought to be useful to terminate episodes early if a satisfactory result could no longer be obtained, because

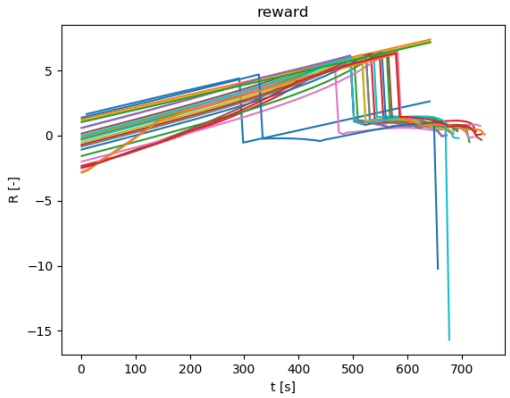


Figure 4-14: Reward vs. time for 25 test episodes after 375 episodes of training with dueling DQN

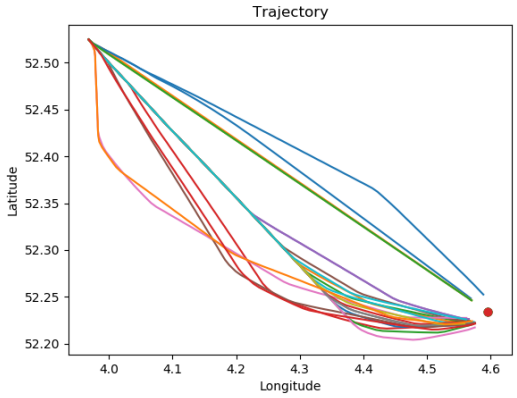


Figure 4-15: Trajectories for 25 test episodes after 375 episodes of training with dueling DQN

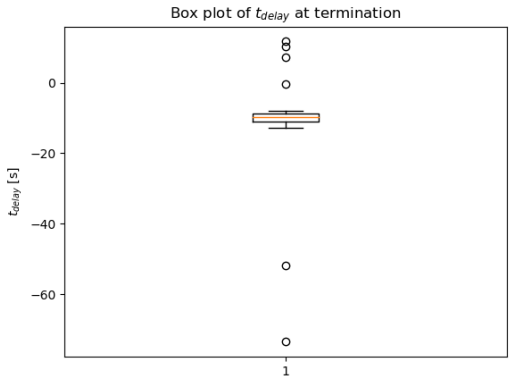


Figure 4-16: t_{delay} at termination for 25 test episodes after 375 episodes of training with dueling DQN

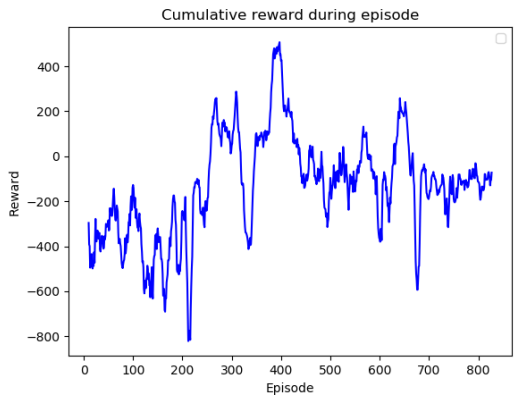


Figure 4-17: Running average of 10 samples for the final reward received during an episode while training with dueling DQN

t_{delay} had become too small. As a result episodes terminate very early while the aircraft is trying to explore the environment, because random actions prevent a good solution. Therefore the exploration of the environment is restricted too much. In the end the agent failed to learn how to fly to the FAF due to lack of proper exploration. It never learned where and how good rewards could be obtained. The end result was immediate 'suicide' by turning instantly to ramp up t_{delay} and quickly terminating the episode as suicide was the fastest way to prevent receiving many negative rewards. Adding a high penalty for terminating an episode through the wrong terminating condition also did not prevent the aircraft from learning 'suicidal' behaviour. This shows that a proper exploration strategy for the environment is of key importance for learning a successful policy. A good exploration strategy in conjunction with poor termination conditions still results in a failure.

Exploration is also accompanied by the term exploitation. Together they form the exploration vs. exploitation dilemma. When is it time to stop exploring and start exploiting the learned policy. As mentioned in the results, there were many issues with the algorithm diverging from an optimal solution, even when a good solution had already been discovered. It was observed that the divergence and the annealing of the ϵ-greedy policy were related in all successful training runs. Every successful

training run diverged again when continuing to train with an exploitation policy after the ϵ was annealed to ϵ_{min}. Exploitation should serve to strengthen the learned policies in the network. However, this did not seem to be the case. This diverging behaviour was also observed when applying the method to a simple 2D inverted pendulum balancing problem when applying DQN, which means it is not an artefact of the environment design in BlueSky.

To Elaborating further on the topic of training instability: This seems to be an issue with applying reinforcement learning that is not mentioned in papers. In a blogpost Google researcher Alex Irpan elaborates on the current status of reinforcement learning [1]. Where supervised learning is a very stable learning process, reinforcement learning is not. The tuning of hyperparameters has become almost almost trivial in training Deep Neural Networks due to the experience empirical results in this research area. Deep reinforcement learning however, is not a stable process at all, as has also become evident in this preliminary analysis. Adding to that the high sample inefficiency of the methods employed, it slows down research. Runs may fail or be successful based on random seed for an identical set of hyperparameters. This also makes reproduction of the work very hard because another implementation of the same algorithm will not produce the same result. To put in this in context for this work. To learn a relatively simple problem of an aircraft flying towards a FAF, a run of 1000 episodes could range from 6-8 wall-clock time of training. This training time mostly to the fact the BlueSky environment is expensive to evaluate. Then there is a high chance that the run fails based on random seed. This makes debugging very slow, because it is hard to determine what the cause of a failure may be.

4-5 Navigation by waypoint selection

In order to provide a more stable learning signal an alternative problem representation for free flight is investigated. Instead of freely navigating through the airspace, the aircraft is on-route to the airport and has to build the actual route from waypoints that are layed out in a cross-track fashion as shown in Figure 4-18. There are multiple reasons for doing so. First of all the flight paths observed in section 4-2 did not represent realistic aircraft flight paths. Secondly, an advantage of abandoning complete free flight is that the agent now has to build a route from flight segments. That means when the agent starts learning, it will always arrive at the FAF, which is not guaranteed in free flight. The aircraft no longer has to learn how to navigate to the FAF in the first place, but rather what the most efficient choice of routing is in the current traffic situation. This provides a more stable reward signal for the agent. It is expected that this will stabilize the learning process by providing a stronger learning signal as compared to 4D trajectory learning in section 4-2.

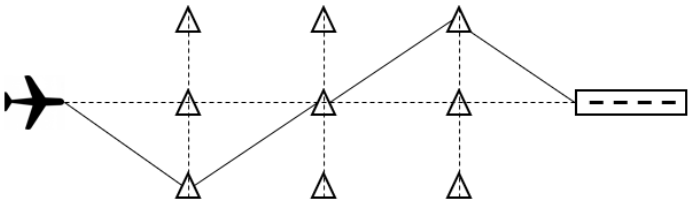


Figure 4-18: Cross track navigation with waypoint

[1] Alex Irpan (Google), 'Deep Reinforcement Learning Doesn't Work Yet', 2018 `https://www.alexirpan.com/2018/02/14/rl-hard.html` [accessed 8 August 2018]

4-6 Experiment description

The observation set $\mathcal{S}$ for the aircraft is a combination of a local observation and a shared observation. The local observation is build up of properties of the aircraft itself, namely the calibrated airspeed, the segment index on which it flies, latitude and longitude. $\mathcal{S} = [V_{cas}, \text{lat}, \text{lon}, \text{lat}_{\text{prev}}, \text{lon}_{\text{prev}}, \text{lat}_{\text{target}}, \text{lon}_{\text{target}}]$. The shared observation is a global encoding of the environment. The shared observation is added to give aircraft more situational awareness in the multi-agent setting. Intuitively this makes sense. It can be compared to two persons standing in a room filled with obstacles. They have to walk towards each other to shake hands, using their own observations and communication. This will be much easier to achieve when the light is on (shared observation) than when the room is pitch black. However, for this experiment the shared observation is not yet included and hence also not yet designed, because only one aircraft will be present in the simulation.

Next is the action set, which is comprised of a waypoint selection and speed selection. In order to keep the optimization simpler there are no separate actions heads for the waypoint selection and action selection. Rather the two are combined into an action set, which can be represented as a matrix for n waypoint options and m calibrated airspeed options. During runtime invalid actions are masked to make sure aircraft has to stay on route to the selected waypoint. This does slow down learning, because aircraft have to learn all 4 trajectories separately.

$$\mathcal{A} = \begin{bmatrix} \text{wp}_1 V_1 & \cdots & \text{wp}_n V_1 \\ \vdots & \ddots & \vdots \\ \text{wp}_1 V_m & \cdots & \text{wp}_n V_m \end{bmatrix}$$

4-7 Method

The BiCNet architecture was selected to solve the problem and to try out communication interface for multiple aircraft. The network architecture provides the mapping from observations to action. The network architecture for BiCNet is displayed in Figure 4-19. The actor is a mapping from state to actions and the critic is value-function and thus a mapping from state and actions to value. First the shared observation is fed through a fully connected layer with ReLu activation to learn a representation for the occupancy of the segments. The local observations are also not fed to the main network directly. Because the segment index would be represented by a one-hot vector it is better to learn an embedding for this parameter. The other input will be normalized at runtime using a all observations collected so far and will be updated each timestep. All the preprocessed observations are then fed through a fully connected layer that shares the same parameters for all aircraft. The sequence of aircraft is then fed to the communication layer, which is a bidirectional recurrent neural network with Long Short Term Memory (LSTM) activations. The output is then again fed through a fully connected layer with shared parameters after the segment and speed action selection is performed using a softmax operation. The critic is a value function and will have the same input as the actor with the addition of the selected actions in the actor and will output the Q-values. The gradient for the network updates will be computed using DDPG for multi-agent systems and a replay memory.

A flow-chart describing the information flow through the simulation is shown in Figure 4-20. The simulation environment is already provided by BlueSky. However, the desired states and actions are not directly available in BlueSky. Therefore a communication interface is created, which translated the action selection into the correct BlueSky commands and uses the states available in BlueSky to deliver compose observations in the correct format. A reinforcement learning plug-in is written for

BlueSky created that interacts with the communication interface. This allows for easy swapping of network architectures and training methods if deemed necessary. Illegal actions must be masked so they cannot be selected. Illegal actions can occur when the aircraft will eventually as invalid otherwise there will always remain a small probability that the action is chosen. Training will be performed with the BiCNet algorithm. The flowchart is accompanied by the algorithm used for the BiCNet architecture which is shown in Algorithm 2.

Algorithm 2: BiCNet algorithm

Initialize actor network and critic network with ξ and θ
Initialize target actor network and target critic network with $\xi' \leftarrow \xi$ and $\theta' \leftarrow \theta$
Initialize replay memory R
for *episode=1, E* **do**
  Initialise a random process $\mathcal{U}$ for action exploration
  Receive initial observation state s^1
  for *t=1,T* **do**
    For each agent i, select and execute action $a_i^t = \mathbf{a}_{i,\theta}(s^t) + \mathcal{N}_t$
    Receive reward $[r_i^t]_{i=1}^N$ and observe new state s^{t+1}
    Store transition $\left\{s^t, [a_i^t, r_i^t]_{i=1}^N, s^{t+1}\right\}$ in R
    Sample random minibatch of transitions $\left\{s_m^t, [a_{m,i}^t, r_{m,i}^t]_{i=1}^N, s_m^{t+1}\right\}_{m=1}^M$ from R
    Compute target value for each agent in each transition using the Bi-RNN:
    for *m=1, M* **do**
      $\hat{Q}_{m,i} = r_{m,i} + \lambda Q_{m,i}^{\xi'}\left(s_m^{t+1}, \mathbf{a}_{\theta'}(s_m^{t+1})\right.$ for each agent i
    end
    Compute critic gradient estimation according to:
    $\Delta\xi = \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{N}\left[(\hat{Q}_{m,i} - Q_{m,i}^{\xi}(\mathbf{s}_m, \mathbf{a}_\theta(\mathbf{s}_m))) \cdot \nabla_\xi Q_{m,i}^{\xi}(\mathbf{s}_m, \mathbf{a}_\theta(\mathbf{s}_m))\right]$
    Compute actor gradient estimation:
    $\Delta\theta = \frac{1}{M}\sum_{m=1}^{M}\sum_{i=1}^{N}\sum_{j=1}^{N}\left[\nabla_\theta \mathbf{a}_{j,\theta}(\mathbf{s}_m) \cdot \nabla_{\mathbf{a}_j} Q_{m,i}^{\xi}(\mathbf{s}_m, \mathbf{a}_\theta(\mathbf{s}_m))\right]$
    and replace Q-value with the critic estimation
    Update the networks based on Adam using the above gradient estimators
    Update target networks:
    $\xi' \leftarrow \gamma\xi + (1-\gamma)\xi'$, $\theta' \leftarrow \gamma\theta + (1-\gamma)\theta'$
  end
end

4-8 Results and Discussion

Unfortunately there are no valid results for this experiment. A mistake had been made in the training method choice with the problem setup. DDPG is not compatible with discrete actions. Therefore the results are not meaningful. The experiment was discontinued without first trying other training algorithms with the BiCNet architecture. More issues emerged with the whole problem setup. Due to the fact that the actual choice of aircraft speed and waypoint selection at different time scales, while both actions were coupled in the action choice this causes noise in the action selection. The problem would have to be modeled as a SMDP problem. This leads to compatibility issues with the BiCNet architecture because. Aircraft do not select actions simultaneously due to the SMDP while BiCNet outputs actions for every aircraft in a forward network pass. This will give issues in the training phase where backpropagation is actually only applicable to one of the actions heads and does provides false gradient information to all the other action heads. A hierarchical model architecture similar as was

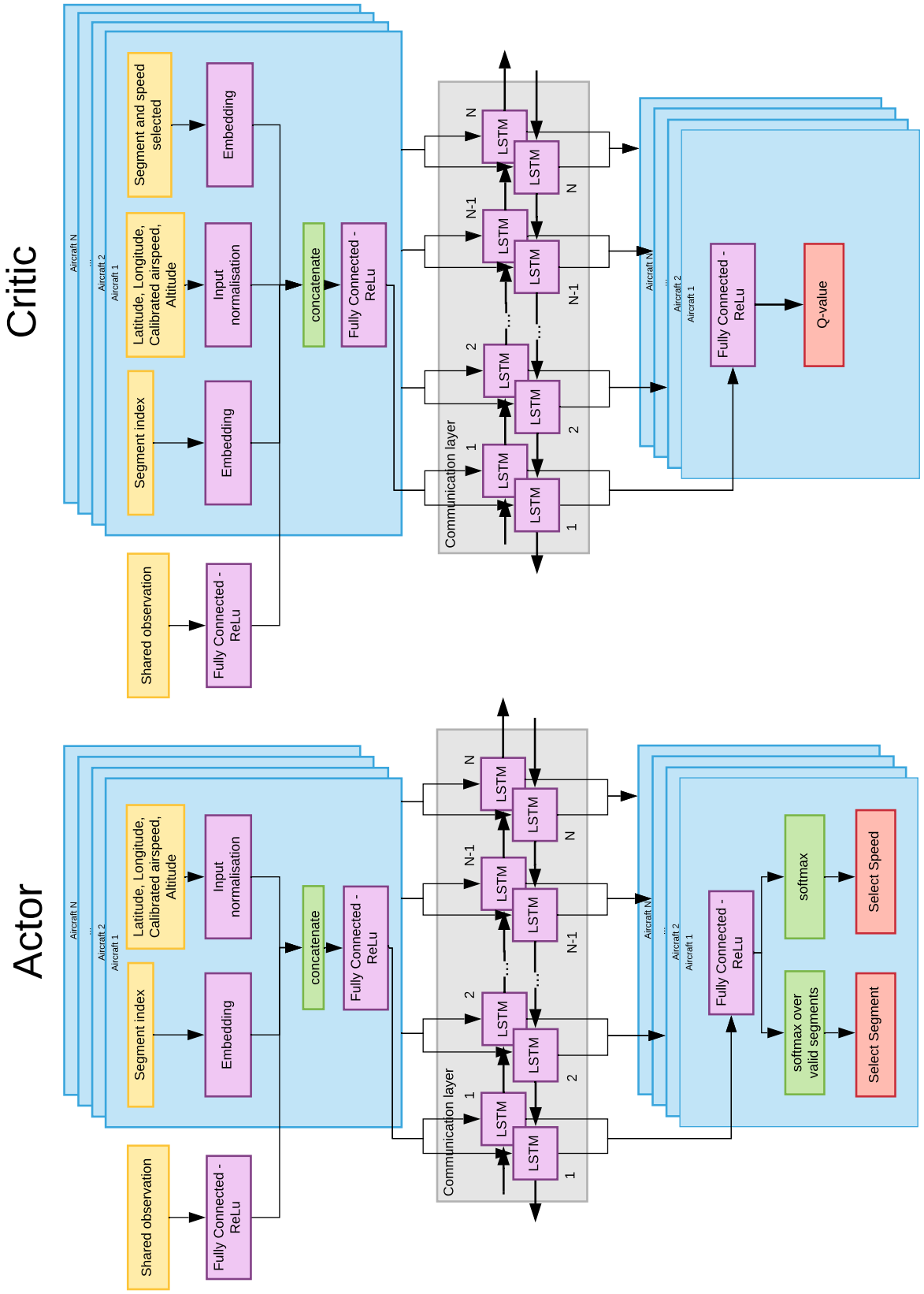


Figure 4-19: Network architecture of the BiCNet implementation in BlueSky to control air traffic simulation

demonstrated by Brittain and Wei (2018) would be more suitable. This however, gives other problems in the multi-agent setting with the variable number of aircraft because a fixed network architecture is used. Therefore navigation by waypoint selection is abandoned for now.

4-9 Conclusion

To conclude the preliminary analysis. The experiments have yielded successful results. Aircraft could learn consistently fly towards the final approach fix while absorbing an assigned delay. The arrival time was generally biased to arriving late in both the DQN approach as well as the Dueling DQN approach. Due to the different hyperparameters, reward function and the stochastic in the simulation the results if both methods cannot be compared directly. The influence of a strong exploration strategy on the end result was very evident. The exploration strategy can speed up learning, or prevent the aircraft from learning anything useful at all. Deep reinforcement learning has many pitfalls and instabilities that are sometimes difficult to avoid.

To come back to the initial question what reinforcement learning techniques are suitable to model the sequencing and spacing of aircraft in the TMA. Some single-agent techniques have been tried that handle a continuous state space and discrete action space. The technique of learning to fly to waypoints is likely not the most suitable way to model the navigation of an aircraft. When looking ahead, the simulation will contain multiple aircraft. It is already hard to find a suitable policy for a single agent. Therefore a new model of the environment should be considered for the final experiment.

Other less-successful representations were explored in the form of navigation by waypoints. The BiCNet architecture was applied here to guide aircraft. However, the BiCNet algorithm, which is based on the DDPG was not compatibly with the problem representation.

In general when applying reinforcement learning, it is good to strip the problem down as much as possible initially. There are so many possible causes for failure that learning and debugging is almost a prohibitive task. Good habits would include implementing the intended learning algorithm and applying it to a known benchmark problem such as balancing an inverted pendulum. In that way the expected behaviour and a working set of hyperparameters are readily available to verify the implementation of the learning algorithm. Next apply it to the environment of a unknown problem. Start with a very basic version to do a basic hyperparameter search and start designing the reward function and expand the problem from there.

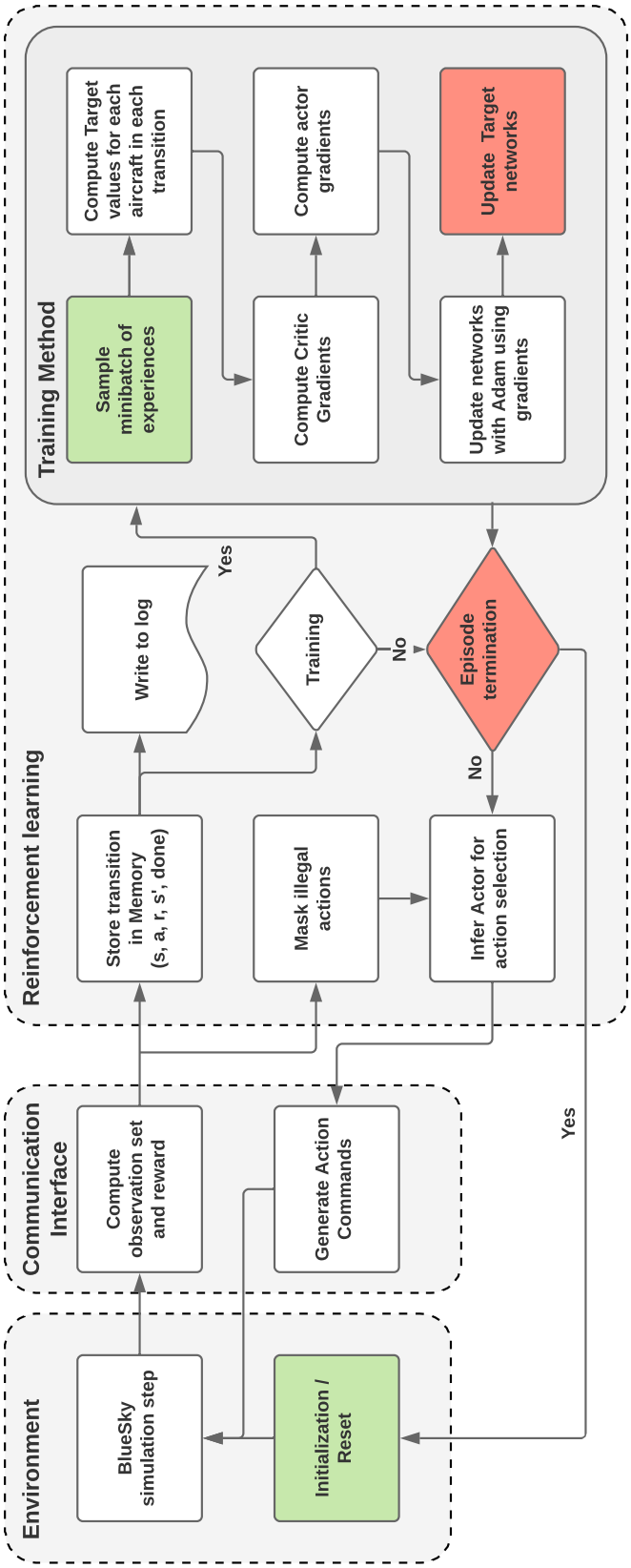


Figure 4-20: Network architecture of the BiCNet implementation in BlueSky to control air traffic simulation

Chapter 5

Thesis proposal

Based on the findings from the preliminary analysis a proposal for further thesis work is constructed. To answer the research question: 'What is the safety and efficiency of the emerging strategies for sequencing and spacing aircraft in the TMA learned by a controller trained using reinforcement learning?', further research must be conducted. The first research sub-question has been answered by the combination of a literature study and a research question. To find an answer to the remaining two research sub-questions an additional experiment must be designed.

To move into the realm of multi-agent simulation some additional representation challenges must be met. First of all, the number of aircraft present at runtime must be variable. Therefore the network architecture must be able to handle this. Previous approaches tried like MADDPG and DQN do not support multi-agent simulations with variable number of agents at runtime. The reasons usually are that every agent will be trained with its own unique policy. When aircraft are added or removed from the simulation this becomes a problem, because the trained data cannot be applied to the new agent. These challenges will be overcome with the use of BiCNet.

5-1 Problem statement

The experiment will be a experiment with multiple aircraft. The air traffic control problem is modelled as a multi-agent reinforcement learning problem. In order to study emerging strategies aircraft will be given the freedom of free flight, but in a slightly different fashion than the 4D trajectory navigation in section 4-2. In this way aircraft are given enough freedom for strategies to emerge. Aircraft will enter the experiment area when entering the Dutch airspace to land at Schiphol airport. The simulation will be run in the BlueSky Open Air Traffic Simulator Hoekstra and Ellerbroek (2016). The goal for the aircraft is to maximize runway throughput while maintaining safe separation and adhering to minimum wake separation criteria using the times in trail criteria. All information available can be used to allow for full observability of the state space.

The observation set $\mathcal{S} = [\mathcal{S}_{\text{local}}, \mathcal{S}_{\text{shared}}]$ for the aircraft is a combination of a local observation and a 'shared observation'. The local observation consists of the relevant states of the aircraft itself. $\mathcal{S}_{\text{local}} = [\text{lat}, \text{lon}, \text{hdg}, \text{qdr}_{\text{target}}, d]$ contains the aircraft latitude, longitude, heading, bearing with respect to the destination and the distance to the destination. The shared observation is designed to create situational awareness for an aircraft by observing the state of other aircraft. Intuitively this makes sense. It can be compared to two persons standing in a room filled with obstacles. They

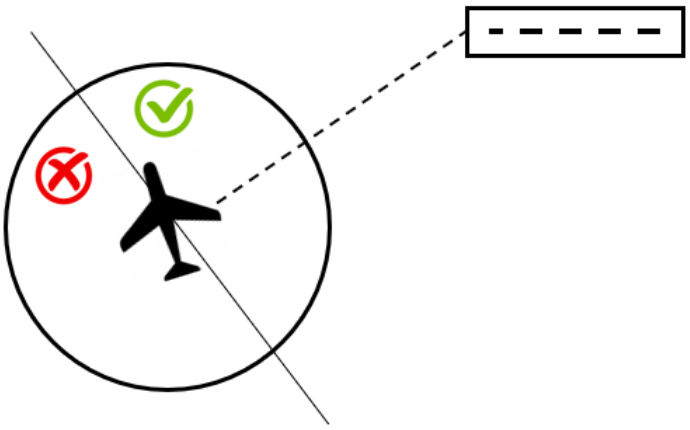


Figure 5-1: Heading range for action selection

have to walk towards each other to shake hands, using their own observations and communication. This will be much easier to achieve when the light is on (shared observation) than when the room is pitch black. Therefore each aircraft observes the following set of states for every other aircraft $\mathcal{S}_{\text{shared}} = [\text{lat}, \text{lon}, \text{hdg}, \text{qdr}, d]$. So the shared observation is a sequence of aircraft states, where every sequence entry contains the latitude, longitude, heading, bearing with respect to the other aircraft and the distance to the other aircraft. The shared observation is almost the same for every aircraft, hence its name.

The action space will only contain a heading selection for each aircraft, thus $\mathcal{A} = [\text{hdg}]$. This choice is made to keep the action space relatively more simple and may be extended in the future. To find a compromise between learnability and freedom of action, the actions are bounded to the range -90 deg and 90 deg with respect to the bearing to the target as shown in Figure 5-1. This forces the aircraft towards the target.

To complete the description of the reinforcement learning method a reward function must be specified. Every aircraft will receive its own individual reward. This is necessary, because it is difficult to assign credit to individual agents' actions based on a single global reward. The individual rewards are a combination of local reward and a global reward. The local reward shall have the following components: loss of separation, progress Global reward: fuel used. The goal is to maximize runway throughput while maintaining a safe separation and efficient flight. Therefore these components will be in the reward function. Wake separation criteria for miles in trail as listed in Table 5-1 must be met as well as minimum spatial separation criteria.

5-1-1 Generating traffic scenarios

A traffic scenario is generated using data from Demand Data Repository II (DDRII) repository provided by EuroControl. This repository contains historical flight data ranging from filed flight plans to the actual flown trajectories. To simulate real world traffic demand historical data filtered for EHAM arrivals is downloaded. A flight consists of a set of flight segments where each flight segment is marked with a start position, start speed, start time, start altitude, end position, end speed, end time, end altitude, the length of the segment and the heading on that segment. For each flight, the segment where the aircraft enters Dutch airspace is found and interpolated to define the starting data of the aircraft.

Table 5-1: RECAN-EU minimum wake separation criteria during approach Rooseleer et al. (2015)

Leader \ Follower		Super Heavy A	Upper Heavy B	Lower Heavy C	Upper Medium D	Lower Medium E	Light F
Super Heavy	A	3 NM	4 NM	5 NM	5 NM	6 NM	8 NM
Upper Heavy	B		3 NM	4 NM	4 NM	5 NM	7 NM
Lower Heavy	C		2.5 NM	3 NM	3 NM	4NM	6 NM
Upper Medium	D						5 NM
Lower Medium	E						4 NM
Light	F						3 NM

5-2 Method

In order to solve this problem, multiple aircraft must be controlled simultaneously with a variable number of aircraft during runtime. The network architecture provides the mapping from observations to action. The network architecture for BiCNet is displayed in Figure 5-2 and has been changed in some location with respect to the architecture proposed in chapter 4. The actor is a mapping from state to actions and the critic is value-function and thus a mapping from state and actions to value. First the shared observation is fed through a fully connected layer with ReLu activation, which has the same weights for every sequence entry in the shared observation. The result is then max-pooled along the aircraft axis to obtained a vector that is fixed in size to be concatenated with the local observation. The local observations are also not fed to the main network directly, but are also fed through a shared fully connected layer with ReLu activation of which the weights are the same for every aircraft. All the preprocessed observations are then fed through a fully connected layer that shares the same parameters for all aircraft. The sequence of aircraft is then fed to the communication layer, which is a bidirectional recurrent neural network with LSTM activations. The output is then again fed through a fully connected layer with shared parameters after the segment and speed action selection is performed with a tanh activation to obtain a value in the range[-1, 1]. The critic is a value function and will have the same input as the actor with the addition of the selected actions in the actor and will output the Q-values.

The gradient for the network updates will be computed using DDPG for multi-agent systems and a replay memory. Training will be performed with the BiCNet algorithm. The flowchart is accompanied by the algorithm used for the BiCNet architecture which is shown in Algorithm 2. A flow-chart describing the information flow through the simulation is shown in Figure 4-20. The simulation environment is already provided by BlueSky. However, the desired states and actions are not directly available in BlueSky. Therefore a communication interface is created, which translated the action selection into the correct BlueSky commands and uses the states available in BlueSky to deliver compose observations in the correct format.

5-3 Verification & Validation

In order to verify the correct implementation Tensorboard will be used, which is a tool included in the TensorFlow library. As verification is not possible in the sense of a traditional verification process. Van Wesel and Goodloe (2017) divides the process into offline and online verification and identifies the following that can be verified:

Offline:

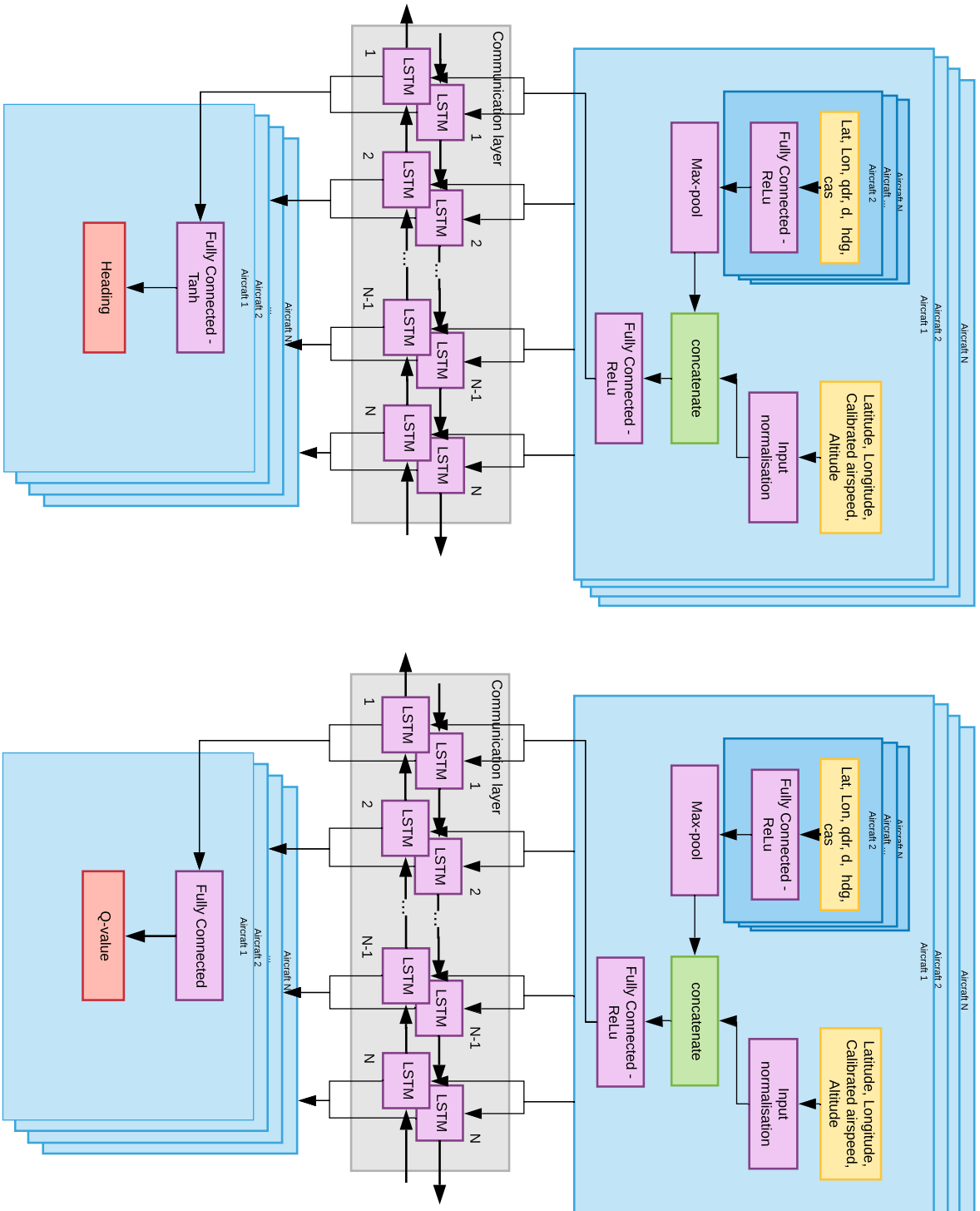


Figure 5-2: Network architecture of the BiCNet implementation in BlueSky to control air traffic simulation

- State to action mappings - Verify the correct input output shape and data type as well as the correct masking of illegal actions.
- Action sequences - Check if the selected actions lead to the correct corresponding state transition.
- Algorithm properties independent of data - Check the convergence properties of the algorithm independently. Proof of the algorithm convergence is already given in the papers.
- Validating assumptions on training data - Check if the variable ranges that are present in the training data reflect what is to be expected.

Online:

- Tensorboard visualisation of weight evolution and variable evolution. The rewards obtained can be visualized live to see training progress. The gradients can be monitored as well to see if the optimization process is healthy. Additionally, the evolution of distribution of the weights can be monitored in Tensorboard using histograms.

Validation of a reinforcement learning is not the same thing as validation of a supervised learning application. Usually, in machine learning applications a training set, a test set and a validation set are used to make sure the learned model fits the problem well. The training set is used to train the network, the test set is used as independent test set to compare with the training result to prevent overfitting the training set. Multiple iterations are done to obtain the same accuracy for the training and test set. Finally the validation set is used as a final test to see how the algorithm will do in the real world as the hyperparameters were tuned to fit both the training and test set well.

Reinforcement learning is different, because it doesn't just use a dataset to train on, but interacts with an environment to learn to maximize an expected reward. The reward is designed to promote desired behaviour. The received reward can thus act as a measure for the performance. However, separate measures can be used to see if the agent actually learned a desired behaviour rather than just a trick to obtain a high reward. The controller is trained on a traffic load for arriving flights to Schiphol. Therefore a number of traffic scenario's adapted from historical flight data must be run tested with the reinforcement learning controller serving as the actual controller. Real flight data can be taken from the DDRII repository from EuroControl. The score of the controller can then be compared with the score obtained from simulating the actual flight data separated by human air traffic controllers as a baseline.

Metrics used are:

- Sequence stability
- Fuel used per aircraft by calculating work done of an aircraft.
- Number of conflicts
- Number of intrusions

5-4 Outcome

The research questions states the interest in the safety and efficiency of the emerging strategies of the controller. Next to that it also interesting to identify some of the strategies to get insight in what the agents have actually learned. The most interesting part is to see how well the controller generalizes to situations it has not seen before. Reinforcement learning is not known for its power to generalize.

It is interesting to see if the controller then generalizes to solve a real-traffic scenario to compare with the baseline or if additional training on a specific scenario is required to achieve better performance. The generalization power is also a good prospective of the feasibility for the implementation of a reinforcement learning controller for eg. airspace design.

To assess the safety and efficiency following will be computed:

- Fuel used per aircraft by calculating work done of an aircraft.
- Number of conflicts
- Number of intrusions
- Sequence stability

Sequence stability is a measure to assess the efficiency and workload of sequencing and spacing and is defined by positional changes in the sequence. The sequence is determined by computing expected arrival time deterministically from the current aircraft state and the optimal trajectory to the runway. Sequence changes can be detected by comparing the expected arrival sequence on multiple snapshots in time to see when the aircraft will change position in the sequence and what caused this, eg. relative speed, entry position, position compared to other agents.

It is interesting to see if the controller then generalizes to solve a real-traffic scenario to compare with the baseline or if additional training on a specific scenario is required to achieve better performance.

5-5 Planning

The progress and planning of the thesis project is elaborated upon. A Gantt chart showing the global progress can be found in Figure 5-3. The final experiment phase has started. The main tasks left are programming the final experiment environment in BlueSky, followed by an experiment phase and analysing and writing the thesis.

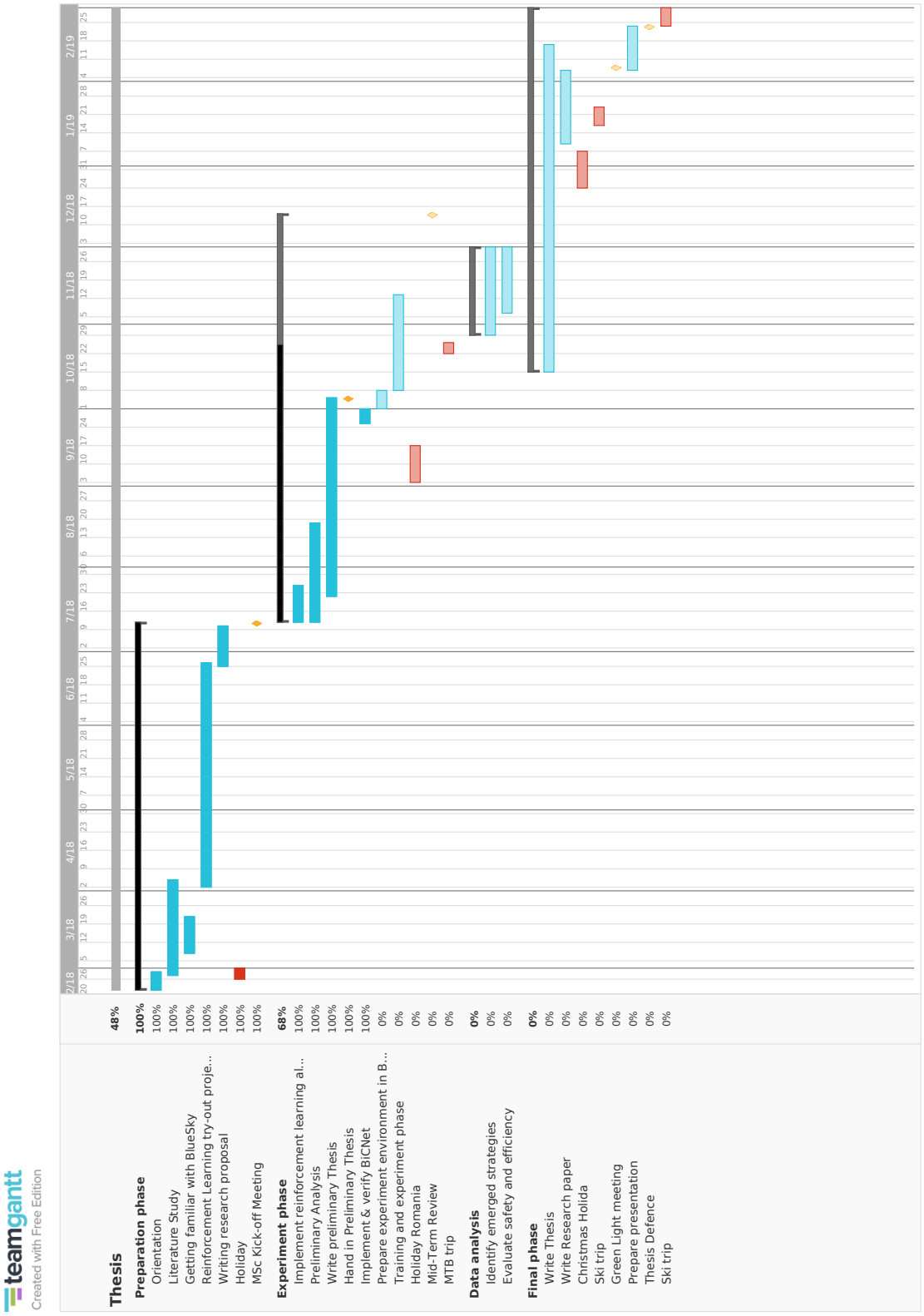


Figure 5-3: Thesis planning

Chapter 6

Conclusion

To apply the recent developments in artificial intelligence to air traffic management a literature study has been conducted where both reinforcement learning as well as automation of air traffic control in the terminal manoeuvring area have been studied. Methods like the DQN

A preliminary analysis was conducted aiming at the complete autonomy of individual aircraft. Using the reinforcement learning methods Deep Q-Networks and Dueling Deep Q-Networks a single aircraft was trained to navigate a 4D trajectory. Policies to control the aircraft were learned successfully, despite a unstable learning process. The Dueling DQN showed faster learning than the DQN. The importance of a properly tuned exploration strategy and reward function became evident through the occurrence of reward hacking and clear improved learning results based on varied exploration rates. Causes of which were the training instability of the training method, the problem set-up, exploration vs. exploitation choices and reward function shaping. Finally a navigation by cross-track waypoint selection has been investigated as well, but is abandoned due to the incompatibility of that problem setup with the desired architectures of the multi-agent network.

A thesis proposal has been made to answer the latter to answer the research question: What is the safety and efficiency of the emerging strategies for sequencing and spacing aircraft in the TMA learned by a controller trained using reinforcement learning? The problem setup is transferred to a multi-agent setting. Due to the observed learning difficulties encountered in the single agent setting a simplified problem is first solved in the multi-agent setting. The aircraft should learn to avoid each other and line up for sequencing and spacing. Safety and efficiency will be measured by sequence stability, number of intrusions, fuel used by aircraft. The results will be compared to a baseline produced from historical flight data, which represents the current efficiency.

Appendix A

DQN Summary

In this appendix all hyperparameters values used in the preliminary analysis experiment runs that have been presented in this report are listed.

Reward function

$$R = (5 + \alpha_d d) + (10 + \alpha_t \Delta t) + \alpha_{hdg} \Delta\psi \tag{A-1}$$

Table A-1: Reward function parameters

α_d	-0.3
α_t	-0.2
α_{hdg}	-0.07

Table A-2: DQN network summary

Layer 1	Dense 24 - Rectified linear unit
Layer 2	Dense 24 - Rectified linear unit
Layer 3	Dense - Linear
Loss function	Mean squared error
Optimizer	Rmsprop

Table A-3: Hyperparameter overview

Memory length	2000
γ	0.98
ϵ_{start}	1.0
ϵ_{min}	0.01
ϵ_{decay}	0.9954
λ	0.001
Batch size	32
c	1000

Appendix B

Dueling DQN summary

Reward function

The reward function in Equation B-1 is build up of three components: Distance, time, and a binary reward based on a time derivative. If the episode terminates without reaching the goal, a penalty is given via the t_{rew}.

$$R = (3 + \alpha_d d) + t_{rew} + dt_{rew}\Delta\psi$$

$$\text{where } t_{rew} = \begin{cases} -100 & \text{if termination penalty} \\ 10 + \alpha_t & \text{if not terminationpenalty} \end{cases}, \text{and} \tag{B-1}$$

$$\text{where } dt_{rew} = \begin{cases} -1 & \text{if } (t > 0 \text{ and } dt > 0) \text{ or if } (t \leq 0 \text{ and } dt > 0) \\ 1 & \text{if } (t > 0 \text{ and } dt \leq 0) \text{ or if } (t \leq 0 \text{ and } dt \leq 0) \end{cases}$$

Table B-1: Reward function parameters

α_d	-0.22
α_t	-0.2
α_{hdg}	-0.07

Table B-2: Dueling DQN network summary

Advantage Layer 1	Dense 128 - Rectified linear unit
Advantage Layer 2	Dense 128 - Rectified linear unit
Advantage Layer 3	Dense 3 - Linear
Advantage Layer 4	Subtract mean Advantage Layer 3
Value Layer 1	Dense 128 - Rectified linear unit
Value Layer 2	Dense 128 - Rectified linear unit
Value Layer 3	Dense 1 - Linear
Add	Advantage Layer 4, Value Layer 3
Loss function	Mean squared error
Optimizer	Adam

Table B-3: Hyperparameter overview

Memory length	2000
γ	0.98
ϵ_{start}	1.0
ϵ_{min}	0.01
ϵ_{decay}	0.99
λ	0.001
Batch size	32
c	1000
Gradient clip value	0.5

Bibliography

a.D. Laud, & Laud, A. (2004). *Theory and application of reward shaping in reinforcement learning* (Doctoral dissertation). Retrieved from https://www.ideals.illinois.edu/bitstream/handle/2142/10797/TheoryandApplicationofRewardShapinginReinforcementLearning.pdf?sequence=2{&}isAllowed=yhttp://scholar.google.com/scholar?hl=en{&}btnG=Search{&}q=intitle:Theory+and+application+of+reward+shapin

Alves, D. P., Weigang, L., & Souza, B. B. (2008, jan). Reinforcement Learning to Support Meta-Level Control in Air Traffic Management. In *Reinforcement learning*. I-Tech Education and Publishing. Retrieved from http://www.intechopen.com/books/reinforcement{_}learning/reinforcement{_}learning{_}to{_}support{_}meta-level{_}control{_}in{_}air{_}traffic{_}management doi: 10.5772/5293

Brittain, M., & Wei, P. (2018). Towards Autonomous Air Traffic Control for Sequencing and Separation - A Deep Reinforcement Learning Approach. , 1–11.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *OpenAI Website*. Retrieved from https://arxiv.org/pdf/1606.01540.pdfhttp://arxiv.org/abs/1606.01540

Crespo, A. M. F., Weigang, L., & de Barros, A. G. (2012). Reinforcement learning agents to tactical air traffic flow management. *International Journal of Aviation Management*, *1*(3), 145. Retrieved from http://www.inderscience.com/link.php?id=45736 doi: 10.1504/IJAM.2012.045736

Durugkar, I. P., Rosenbaum, C., Dernbach, S., & Mahadevan, S. (2016). Deep Reinforcement Learning With Macro-Actions. *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*, 2094–2100. Retrieved from https://arxiv.org/pdf/1606.04615.pdfhttp://arxiv.org/abs/1606.04615

Hoekstra, J. M., & Ellerbroek, J. (2016). BlueSky ATC Simulator Project: an Open Data and Open Source Approach. In *7th international conference on research in air transportation* (pp. 1–8). Retrieved from http://www.icrat.org/icrat/seminarContent/2016/papers/5/ICRAT{_}2016{_}paper{_}5.pdf

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *ICLR 2016*. Retrieved from https://arxiv.org/pdf/1509.02971v2.pdfhttp://arxiv.org/abs/1509.02971 doi: 10.1561/2200000006

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *NIPS'17 Proceedings of the 31th International Conference on Neural Information Processing SystemsProceedings*, 6379—-6390. Retrieved from https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdfhttp://arxiv.org/abs/1706.02275

Man, L. (2015). An agent-based approach to automated merge 4D arrival trajectories in busy terminal maneuvering area. In *Procedia engineering* (Vol. 99, pp. 233–243). Retrieved from www.sciencedirect.com doi: 10.1016/j.proeng.2014.12.531

Melo, F. S., Meyn, S. P., & Ribeiro, M. I. (1997). An analysis of reinforcement learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690. doi: 10.1109/9.580874

Menda, K., Chen, Y. C., Grana, J., Bono, J. W., Tracey, B. D., Kochenderfer, M. J., & Wolpert, D. (2018). *Deep Reinforcement Learning for Event-Driven Multi-Agent Decision Processes.* Retrieved from https://arxiv.org/pdf/1709.06656.pdfhttp://arxiv.org/abs/1709.06656 doi: 10.1109/TITS.2018.2848264

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning.* Retrieved from https://arxiv.org/pdf/1312.5602v1.pdf

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*. Retrieved from https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf doi: 10.1038/nature14236

Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., ... Deepmind, G. (2016). Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd international conference on machine learning* (pp. 1928–1937). PMLR. Retrieved from https://arxiv.org/pdf/1602.01783.pdf

Narvekar, S. (2017). *Curriculum Learning in Reinforcement Learning* (Tech. Rep.). Retrieved from http://www.intplay.com/uploadedFiles/Game

Nikoleris, T., Erzberger, H., Paielli, R. A., & Chu, Y.-C. (2014). Autonomous System for Air Traffic Control in Terminal Airspace. In *14th aiaa aviation technology, integration and operations conference.* Atlanta. Retrieved from https://arc.aiaa.org/doi/pdf/10.2514/6.2014-2861 doi: 10.2514/6.2014-2861

Pineda, F. J. (1987). Generalisation of backpropagation to recurrent and higher order neural networks. *Physic Review Letter, 18*, 2229–2232. Retrieved from https://papers.nips.cc/paper/67-generalization-of-back-propagation-to-recurrent-and-higher-order-neural-networks.pdf doi: 10.1103/PhysRevLett.59.2229

Rooseleer, F., Treve, V., & Phythian, D. (2015). *RECAT-EU European Wake Turbulence Categorisation and Separation Minima on Approach and Departure* (Tech. Rep.). EUROCONTROL. Retrieved from www.eurocontrol.int

Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Tech. Rep.). Cambridge, England: Cambridge University Engineering Department. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.2539{&}rep=rep1{&}type=pdf doi: 10.1.1.17.2539

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., & Abbeel, P. (2016). HIGH-DIMENSIONAL CONTINUOUS CONTROL USING GENERALIZED ADVANTAGE ESTIMATION. *ICLR*. Retrieved from https://arxiv.org/pdf/1506.02438.pdf

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. Retrieved from https://arxiv.org/pdf/1707.06347.pdfhttp://arxiv.org/abs/1707.06347

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. Retrieved from https://pdfs.semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdfhttp://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=650093 doi: 10.1109/78.650093

Sharma, S., & Taylor, M. E. (2012). Autonomous Waypoint Generation Strategy for On-Line Navigation in Unknown Environments. In *Ieee/rsj international conference on intelligent robots and systems.* Vilamoura. Retrieved from http://www.reflexxes.ws/iros2012ws/Paper{_}12.pdf

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic Policy

Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 387–395. Retrieved from http://proceedings.mlr.press/v32/silver14.pdf

Singh, S. P., & Sutton, R. S. (1996). Reinforcement Learning with Replacing Eligibility Traces. *Machine Learning*, *22*, 123–158. Retrieved from https://link.springer.com/content/pdf/10.1007{%}2FBF00114726.pdf

Sukhbaatar, S., Szlam, A., & Fergus, R. (2016). Learning Multiagent Communication with Backpropagation. In *Nips'16 proceedings of the 30th international conference on neural information processing systems* (pp. 2252–2260). Retrieved from https://arxiv.org/pdf/1605.07736.pdfhttp://arxiv.org/abs/1605.07736

Sutton, R. S., & Barto, A. G. (1998). Reinforcement Learning: An Introduction. *MIT Press, Cambridge, MA, A Bradford Book*. doi: 10.1109/TNN.1998.712192

Sutton, R. S., Mcallester, D., Singh, S., & Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, 1057–1063. Retrieved from https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf doi: 10.1.1.37.9714

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*, 181–211. Retrieved from http://www-anw.cs.umass.edu/{~}barto/courses/cs687/Sutton-Precup-Singh-AIJ99.pdf

Tumer, K., & Agogino, A. (2007). Distributed Agent-Based Air Traffic Flow Management. In *Aamas* (pp. 255:1—-255:8). Honolulu: ACM. Retrieved from http://delivery.acm.org/10.1145/1330000/1329434/a255-tumer.pdf?ip=145.94.178.168{&}id=1329434{&}acc=ACTIVESERVICE{&}key=0C390721DC3021FF.512956D6C5F075DE.4D4702B0C3E38B35.4D4702B0C3E38B35{&}{_}{_}acm{_}{_}=1520330595{_}3dbf17a224fef9446dc0e78ba931bac7 doi: 10.1145/1329125.1329434

Van Wesel, P., & Goodloe, A. E. (2017). *Challenges in the Verification of Reinforcement Learning Algorithms* (Tech. Rep.). Retrieved from http://www.sti.nasa.gov

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-learning. In *Proceedings of the thirthieth aaai conference on artificial intelligence* (pp. 2094–2100). Retrieved from http://arxiv.org/abs/1509.06461 doi: 10.1016/j.artint.2015.09.002

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... Blizzard, R. T. (2017). StarCraft II: A New Challenge for Reinforcement Learning. Retrieved from https://arxiv.org/pdf/1708.04782.pdf

Wang, Z., de Freitas, N., & Lanctot, M. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *arXiv*(9), 1–16. Retrieved from https://arxiv.org/pdf/1511.06581.pdfhttp://arxiv.org/abs/1511.06581 doi: 10.1109/MCOM.2016.7378425

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards* (Doctoral dissertation, University of Cambridge England). Retrieved from http://www.cs.rhul.ac.uk/{~}chrisw/new{_}thesis.pdf

Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, *8*, 229–256. Retrieved from https://link.springer.com/content/pdf/10.1007{%}2FBF00992696.pdf

Zuñiga, C., Delahaye, D., & Piera, M. A. (2011). Integrating and sequencing flows in terminal maneuvering area by evolutionary algorithms. In *Aiaa/ieee digital avionics systems conference - proceedings* (Vol. ISBN, pp. 1–32). IEEE. Retrieved from https://hal-enac.archives-ouvertes.fr/hal-00912819/document doi: 10.1109/DASC.2011.6096177

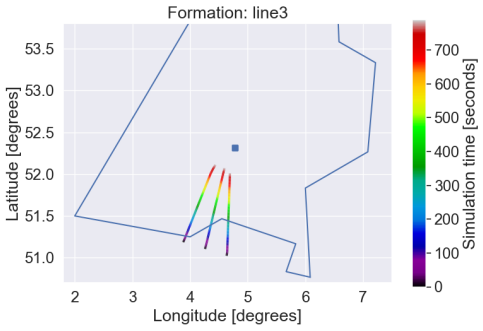
Part III

Appendix

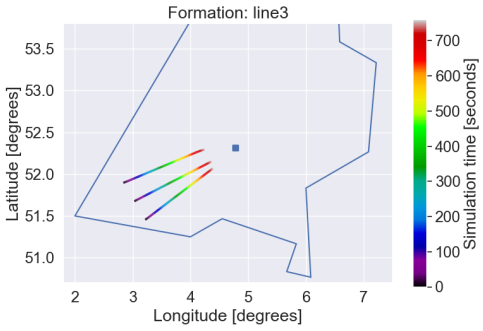
Appendix A

Trajectories experiment 3

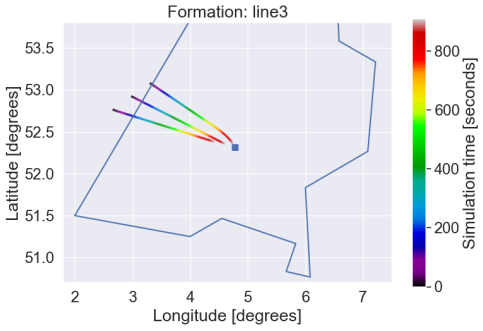
This appendix shows learned trajectories sampled from the set of test scenario's for experiment 3 after 30740 episodes of training.



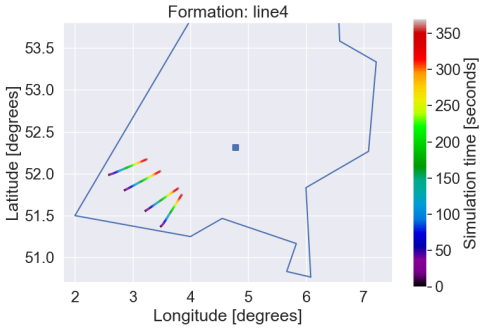
(a) 3 aircraft in line-shaped initialization



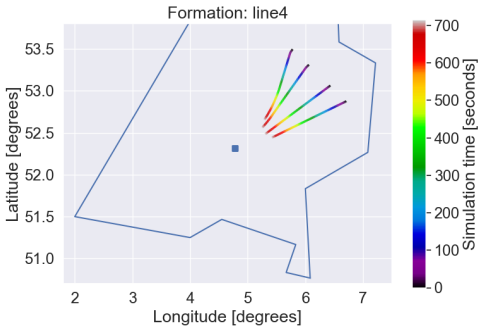
(b) 3 aircraft in line-shaped initialization



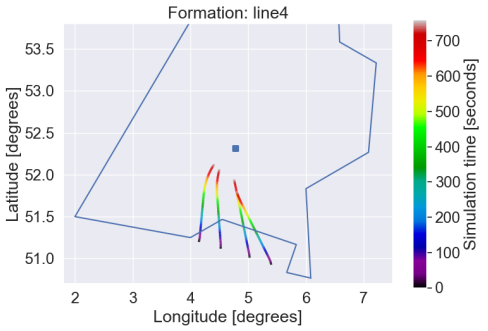
(c) 3 aircraft in line-shaped initialization



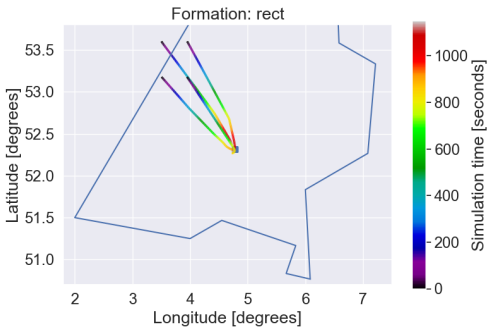
(d) 4 aircraft in line-shaped initialization



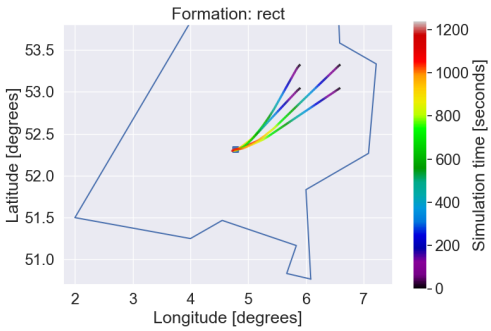
(e) 4 aircraft in line-shaped initialization



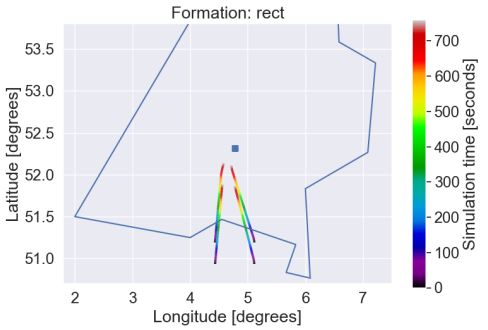
(f) 4 aircraft in line-shaped initialization



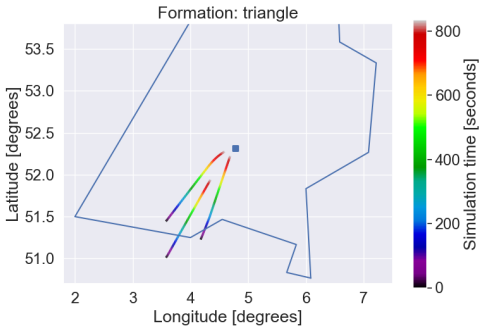
(g) Rectangular shaped initialization



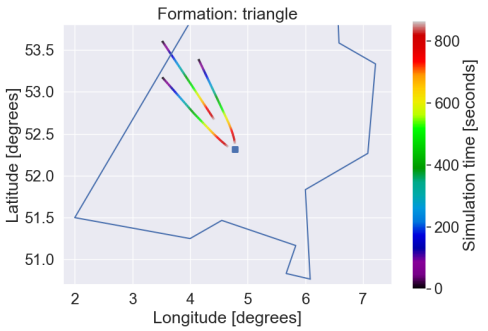
(h) Rectangular shaped initialization



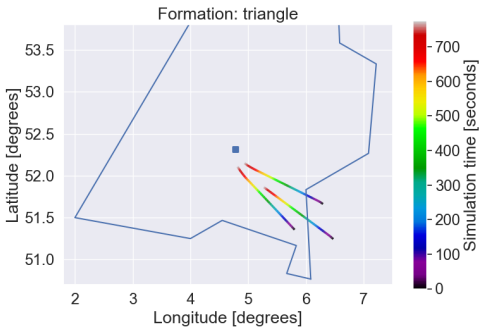
(i) Rectangular shaped initialization



(j) Triangular shaped initialization



(k) Triangular shaped initialization



(l) Triangular shaped initialization

Figure A-1: Test trajectories for experiment 3 after 30740 episodes of training.