UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Reinforcement Learning Approach for Aircraft Profile Optimization

Author:
Mykhailo SHAKHOV

Supervisor:
PhD Taras FIRMAN

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Reinforcement Learning Approach for Aircraft Profile Optimization

by Mykhailo SHAKHOV

Abstract

This project is aimed for using RL approaches for flight planning problem in static and dynamic weather environments. Classical solutions are mostly based on solving non-linear optimization problem with additional non-linear constraints. This kind of approaches are complex, slow and not always can find optimal solution of the initial problem. Current research is dedicated to testing different RL methods that can replace classical ones with at least the same performance.

Acknowledgements

I want to thank my supervisor Taras FIRMAN for generating great ideas and always helping and guiding me in every situation.

I also want to thank the Ukrainian Catholic University for teaching, guiding me, and making me a human.

Contents

List of Figures

List of Abbreviations

RL	**R**einforcement **L**earning
DL	**D**eep **L**earning
DRL	Deep Reinforcement learning
DQN	**D**eep **Q** **N**etworks
DDQN	**D**ouble **D**eep **Q** **N**etworks
TD	**T**emporal **D**ifference
CNN	**C**onvolutional **N**eural **N**etwork
ANN	**A**rtificial **N**eural **N**etwork
IAS	**I**ndicated **A**ir **S**peed
TAS	**T**rue **A**ir **S**peed
TOP	**T**op **O**f **C**limb
TOD	**T**op **O**f **D**escent
WAS	**W**ind **C**orrection **A**ngle
CWA	**C**ourse **W**ind **A**ngle

Dedicated to my loving parents, grandmother, and brother

Chapter 1

Introduction

1.1 Motivation

The Wright brothers did the first powered flight on December 17, 1903. Approximately 17 years after this, the flight center was created. Since then, flight management tasks have appeared. Route planning problem is one of them, and besides of modern world of innovation, technologies it is not perfect. The ones of the leading flight planning problems are reducing the fuel consumption and compliance of flights with Air Traffic Control requirements. Air Traffic Control service monitors the safety of the flight. It controls air traffic, weather conditions, etc. Many researchers have been looking for different solutions for this task for years. Some of them are trying to solve it with genetic algorithm [1], and some of them treat flight planning as a graph-based problem [2]. These approaches are not perfect; if some can find the solution close to optimal, they require a lot of computing resources and time. That's why such kinds of problems are still open to fresh approaches.

The main problems with such existing approaches are that they are defined for static environment. It means, that all the weather conditions and winds are predicted beforehand. But they are not always precise, so, if they change while flight, the algorithm will recalculate optimal solution. That's why we propose RL approach. It can describe the actions even if the weather conditions will change. The goal of this project is to train RL agents on static environment, compare results with other approaches, and check the performance on the dynamic one.

1.2 Project Structure

The second chapter describes the main concepts of RL and its broad base. The third one represents the main approaches related to this problem and the paper that can help build the environment. The fourth chapter is about environment building, and the next one describes the RL agents that are used for this work. After that, the results are described and represented in visualizations. The last chapter represents the conclusions and next steps of this problem.

Chapter 2

Background Information

2.1 RL basics

Reinforcement Learning is one of the main areas of Machine Learning. It is about the agent interacting with the environment that it knows nothing about by tries and errors. The agent is trying to influence the environment by doing actions, and the environment gives feedback for this. The goal of RL algorithms is to teach an agent to do such actions that maximize cumulative reward. It is similar to how human beings learn to do something. Several main elements belong to the reinforcement learning system:

MDP can formalize such kinds of problems. MDPs are a classical formalization of sequential decision-making. In this process, actions can be influenced not only by current rewards but also by subsequent situations. Thus it can be used to maximize the cumulative reward – the main goal of RL algorithms. The decision-maker is called an agent, and the thing that it interacts with is called the environment.

The RL algorithms have some basic concepts that they can manipulate:

- **State**: is a description of the state of the environment. For example, it can be the location of chess pieces on the board, or it can be an RGB image of the game.

- **Action Space**: is a set of actions that an agent can do. Action space can be discrete or continuous. For example, discrete space can be the movements of a character in some games; continuous space can describe the robot's control in the physical world.

- **Policy**: is a function or rule that helps an agent to decide what action to do being in a certain state. A policy can be deterministic or stochastic. The deterministic policy defines what action an agent should do, being in a specific state, while the stochastic policy returns the probability of taking action given the state.

- **Reward, reward function**: reward function takes as arguments the current state and some action from action space and returns a numerical value called reward. This value is feedback from the environment to do some action given state. This feedback can be good, bad, or neutral. Maximization of cumulative reward is the goal of RL algorithms, so the design of this function is one of the core principles of projecting RL solutions. For example, in chess, we can design our reward function in such a way that for taking opponent's pieces, we get a positive reward and a bad - one for losing our own ones. But the main task of a chess game is to get to mate. So, we need to assign a much bigger reward for checkmate.

2.2 MDP

As it was said in the previous section, in RL environment the agent makes some action a_t, being in certain state s_t and it moves to next state s_{t+1}. Such process can be called transition function. The general transition can be written by following expression [12]:

$$s_{t+1} \sim P(s_{t+1}|(s_0, a_0), ..., (s_t, a_t))$$

This means that when the agent is in state s_t next state s_{t+1} can be sampled with some probability distribution. As we can see, this transition depends on all previous state-action pairs that the agent has experienced. This makes the environment complicated and not practical, so we assume that this transition depends only on the current state-action pair:

$$s_{t+1} \sim P(s_{t+1}|(s_t, a_t))$$

With such assumption that is called Markov Property the transition function turns into MDP. To finally formulate MDP as RL problem we introduce 4-tuple S (set space), A (action space), $R(s_t, a_t, s_{t+1})$ (reward function), $P(s_{t+1}|s_t, a_t)$ (transition probability). Also, as it was said in previous section, the main goal of RL approaches is to maximize cumulative reward. So we define it the next way:

$$R((s_0, a_0, s_1), ..., (s_{N-1}, a_{N-1}, s_N)) = \sum_{t=0}^{N} \gamma^t r_t$$

$\gamma \in [0, 1]$ is called the discount factor, and it is a very important thing in RL. It shows that the reward of current state action is more important than the reward that the agent gets in the future. It is also can be possible that we have an infinite number of episodes, so if we set $\gamma < 1$, then the sum will be bounded anyway, so it is also helpful for math.

2.3 Value-Based Algorithms

Value-Based algorithms estimate state-action pair using value functions: $V(s)$ and $Q(s, a)$. The task of an agent is to learn such functions. These functions are used to define the policy, which means that the agent will choose action with the help of these functions. We can define these functions as the expectation of the cumulative discount reward. See formulas [9]:

$$V^{\pi}(s) = \mathbb{E}_{s_0=s, \tau\sim\pi}(\sum_{t=0}^{N} \gamma^t r_t)$$

$$Q^{\pi}(s, a) = \mathbb{E}_{s_0=s, a_0=a, \tau\sim\pi}(\sum_{t=0}^{N} \gamma^t r_t)$$

In the case of $V(s)$ value function tells the agent how good it is to be in the state $V(s)$ under the policy π, and the $Q(s, a)$ defines the value of taking action a being in state s under policy π

2.3.1 TD Learning

Now we need to somehow estimate these values. This can be done by TD Learning. The TD learning algorithm is one of the fundamental algorithms of RL. In TD learning, we update the value function using previous and current states:

$$V(s_t) = V(s_t) + \alpha(r_t + \gamma V(s_{t+1}) - V(s_t))$$

Where α is the learning rate, important for function convergence. In this case, we estimate value function, and such approach is called TD prediction.

2.3.2 Q Learning

TD prediction is used for estimation value function, but the approach to optimizing it is called TD control. One of the algorithms of TD control is called Q-Learning - it is an off-policy RL algorithm and one of the most famous approaches for agent learning at all. As its name says, it is based on optimizing Q-function. The rule for updating is defined:

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

There is also one thing that should be mentioned. When the agent selects an action, he chooses the best action due to policy. It does not always provide the optimal result. Here is why we need such a thing as an exploration-exploitation trade-off. When the agent always takes maximum values, he can never find some solution that can be optimal since the policy does not tell the agent to do this. If it takes a lot of exploration, it can waste a lot of time doing unnecessary things. That is why we need to balance these two concepts.

One of the approaches for solving this problem is ϵ-greedy algorithm. It introduces some variable ϵ that is the probability of making random actions. In most cases, we first set this value to 1 so that the agent can explore more new actions. And after every step, we reduce it by some rule so that the agent can do more exploration and not forget about exploitation.

We can define a general Q-Learning algorithm with ϵ-greedy policy:

Algorithm 1: Q Learning

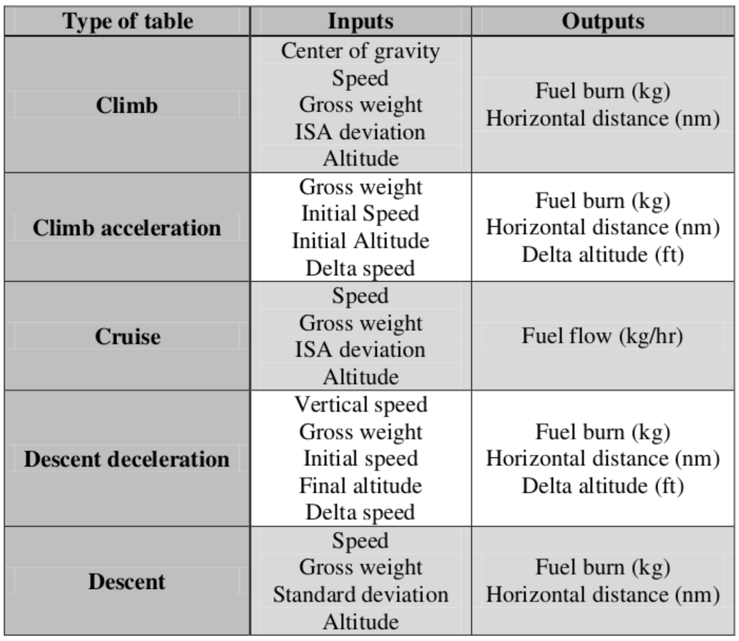
```
Result: Write here the result
$InitializeQ(s, a) forall s \in S, a \in A$;
for each episode do
    Init S;
    for each step in episodes do
        Select A due to epsilone greedy policy;
        observe R, S_{next};
        $Q(S, A) = Q(S, A) + \alpha(R + \gamma \max_a Q(S_{next}, a) - Q(S, A))$;
        S = S_{next};
        if S is terminal then
            go to next episode;
        end
    end
end
```

Chapter 3

Related Works

3.1 OpenAP

OpenAP is an open-source aircraft performance model for air transportation studies and simulations [11]. In order to build the environment of considered problem we need to know the fuel consumption. Many works that suggest different approaches to solve such kinds of problems use PDB – numeric model of each aircraft. The example of PDB input and outputs for Airbus A310 is on Figure 3.1 The main parameter



Type of table	Inputs	Outputs
Climb	Center of gravity Speed Gross weight ISA deviation Altitude	Fuel burn (kg) Horizontal distance (nm)
Climb acceleration	Gross weight Initial Speed Initial Altitude Delta speed	Fuel burn (kg) Horizontal distance (nm) Delta altitude (ft)
Cruise	Speed Gross weight ISA deviation Altitude	Fuel flow (kg/hr)
Descent deceleration	Vertical speed Gross weight Initial speed Final altitude Delta speed	Fuel burn (kg) Horizontal distance (nm) Delta altitude (ft)
Descent	Speed Gross weight Standard deviation Altitude	Fuel burn (kg) Horizontal distance (nm)

FIGURE 3.1: PDB inputs and outputs. Taken from [1]

we need is fuel flow for the cruise since our goal is to optimize fuel consumption. The main problem with this is that it is not open source.

OpenAP helps with this problem. There are four main components in the OpenAP Model: aircraft and engine properties, kinematic performances, dynamic performances, and utility libraries. There is also an open-source library written in Python that includes all the components and features noted in the paper.

- Aircraft and engine properties. This component includes main aircraft and engine parameters that can be used for calculating values in the environment. OpenAP library includes a database that contains approximately 25 different types of aircraft and around 400 different engines.

- Kinematic model. The library also contains a kinematic performance database, where data is collected using a data-driven approach. It divides flight into 7 phases: Takeoff, initial climb, cruise, descent, final approach, and landing. Deals with parameters such as aircraft speed, altitude, distances, etc. It can be essential since it provides us with useful parameters of speed. See Figure 3.2. Especially cruise altitude and Mach number ranges. These parameters will be used in environment building.

- Dynamic Performance. The dynamic performance components is needed manipulation with such parameters as forces and mass. The authors emphasize that the main challenge of constructing dynamic models is the lack of open data. OpenAP accomplishes the goal by using models from the literature and available flight data [11]. This component may be the most important because it can calculate the fuel flow, and using it, we can find fuel consumption.

- Utility library. A utility library is quite a nice feature. It includes a flight phase library, aeronautical calculations, navigation database, etc. It is not used in this work.

Takeoff	V_{lof}	liftoff speed	m/s
	d_{tof}	takeoff distance	km
	$\bar{a}_{\text{tof}}$	mean takeoff acceleration	m/s^2
Initial climb	$V_{\text{cas,ic}}$	calibrated airspeed	m/s
	$V_{S\,\text{ic}}$	vertical rate	m/s
Climb	$R_{\text{top,cl}}$	range to the top of climb	km
	$h_{\text{cas,cl}}$	constant CAS crossover altitude	km
	$V_{\text{cas,cl}}$	constant CAS	m/s
	$V_{S\,\text{cas,cl}}$	vertical rate during constant CAS climb	m/s
	$h_{\text{mach,cl}}$	constant Mach climb crossover altitude	km
	M_{cl}	constant Mach number	-
	$V_{S\,\text{mach,cl}}$	vertical rate at constant Mach climb	m/s
	$V_{S\,\text{precas,cl}}$	vertical rate before constant CAS climb	m/s
Cruise	R_{cr}	cruise range	km
	$h_{\text{init,cr}}$	initial cruise altitude	km
	h_{cr}	cruise altitude	km
	M_{cr}	cruise Mach number	-
Descent	$R_{\text{top,de}}$	range from the top of descent	km
	M_{de}	constant Mach number	-
	$h_{\text{mach,de}}$	constant Mach descent crossover altitude	km
	$V_{S\,\text{mach,de}}$	vertical rate at constant Mach descent	m/s
	$V_{\text{cas,de}}$	constant CAS	m/s
	$h_{\text{cas,de}}$	constant CAS crossover altitude	km
	$V_{S\,\text{cas,de}}$	vertical rate at constant CAS descent	m/s
	$V_{S\,\text{postcas,de}}$	vertical rate after constant CAS descent	m/s
Final approach	$V_{\text{cas,fa}}$	calibrated airspeed	m/s
	$V_{S\,\text{fa}}$	vertical rate	m/s
	$\angle_{\text{fa}}$	path angle	deg
Landing	V_{tcd}	touchdown speed	m/s
	d_{lnd}	braking distance	km
	$\bar{a}_{lnd}$	mean braking deceleration	m/s^2

FIGURE 3.2: Performance parameter in OpenAP kinematic model. It was taken from [6]

3.2 Aircraft Profile Optimization using Genetic Algorithms

The Aircraft Profile Optimization problem can be described as a Non-Linear Constrained Optimization problem - the problem where the objective function or some of the constraints are non-linear. In such problem, the main purpose is to select such decision variables from feasible region x_1, x_2, x_3 for the purpose of maximizing or minimizing objective function:

$$f(x_1, x_2, ..., x_n)$$

Feasible regions can be limited by the set of constraints. So, we can formalize this problem:

$$\text{Maximize/Minimize } f(x_1, x_2, ..., x_n)$$

due to constraints:

$$c_1(x_1, x_2, ..., x_n) \leq b_1$$

$$\vdots$$

$$c_n(x_1, x_2, ..., x_n) \leq b_n$$

Among such works, the authors in the paper [1] are optimizing both horizontal and vertical profiles. Generally speaking, they define objective function as a function of fuel consumption. In that case, decision variables are speed, altitude, current trajectory, etc. And the constraints can be the altitude boundaries and arrival time.

Some values can be discrete, like chosen trajectory or altitude step. In this case, we have a Mixed-Integer Non-Linear Constrained Optimization problem. There are a lot of approaches for solving such kinds of problems, and one of them is a genetic algorithm. It is a search heuristic that is routinely used to generate useful solutions to optimization and search problems [6]. It takes inspiration from natural evolution, and it uses such techniques as crossover, mutation, selection, and inheritance. It is a general algorithm for finding global optimum, so it also can solve the Mixed-Integer Non-Linear Constrained Optimization problem.

Chapter 4

Environment Building

4.1 Airspeed

Before building an aircraft model, we need to understand some concepts.

- Altitude is the vertical distance of the aircraft from sea level.
- True Air Speed (TAS) is the actual speed of aircraft relative to the air. And this measure is exactly what we need to calculate travel time and fuel consumption. This value can be calculated in different ways, using other types of speed.
- Indicated Airspeed (IAS) is measured from outside air pressure using a pitot tube. And we can calculate TAS from IAS. One of the main problems of estimating TAS using IAS is that air has a lower density at higher altitudes. Therefore, EAS is not close to TAS there, and that's where Mach number comes for help.
- Mach number is the ratio of TAS due to the speed of sound.
- Ground speed is the measure that we directly use to calculate the flight time. Ground speed is true Airspeed with the influence of winds.
- Crossover altitude is the altitude where TAS calculated from IAS and TAS calculated from Mach number are equal. It is needed in our work because we use IAS to calculate TAS at altitudes less than crossover one and Mach number high than the crossover altitude.

4.2 Wind Model

The next thing that we should care about is the wind model since winds affect horizontal speed and, indirectly - fuel flow. Therefore in our case, travel time and fuel consumption depend on them.

4.2.1 Data collection

In this work we use weather real-time predictions collected from Global Deterministic Forecast System (GDPS) [5]. It is dataset on 1500x751 latitude-longitude grid, see Figure 4.1. There is data for ten days forecast with a period of 3 hours. There is available information about the atmosphere on 28 isobaric levels from 1015 to 50 that we can convert to altitude. There is also included information about wind magnitude and direction.

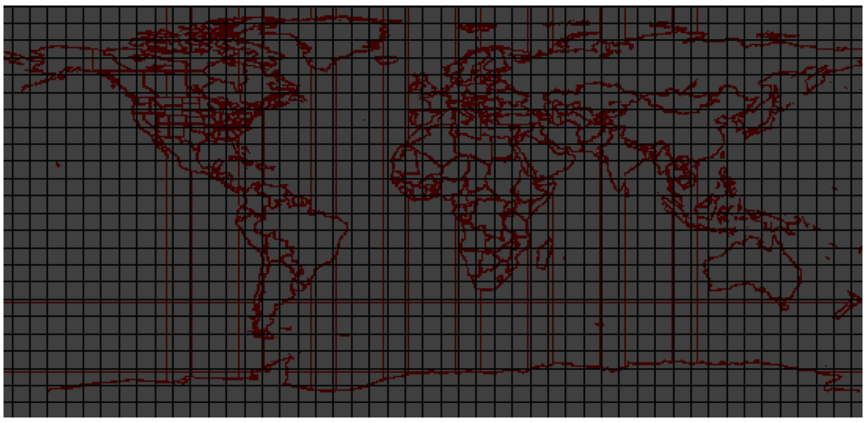


FIGURE 4.1: GDPS on a 25 km full-resolution Lat-Lon grid. Taken from [5]

4.2.2 Data interpolation

Wind data is available on only certain points, but we may have a need to get wind speed that is located between two available points. This problem can be solved by interpolation. The wind direction and magnitude is interpolated between every two consecutive waypoints and also between two related altitudes where wind data is available. The idea is quite similar to the one that is used in [1]. See Figure 4.2

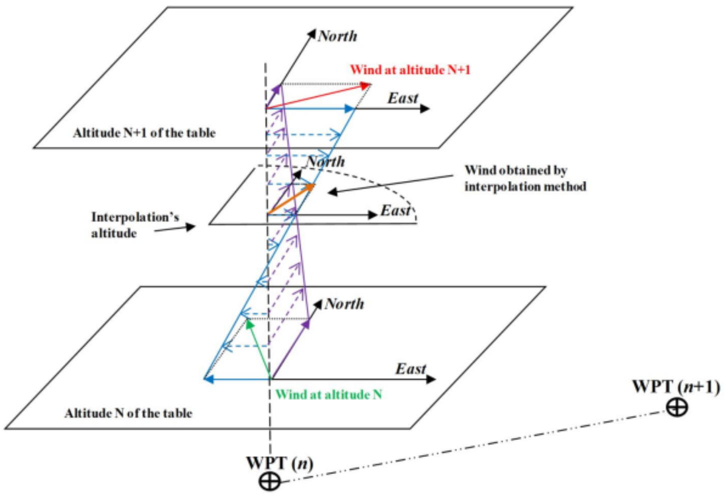


FIGURE 4.2: Wind interpolation. It wsa taken from [1]

4.2.3 Wind triangle

We consider groundspeed to be horizontal speed. Let's check the way to extract groundspeed from TAS and wind speed. It can be done using a wind triangle. One can see the general idea in the Figure 4.3

From [1] we have:

$$\overrightarrow{Groundspeed} = \overrightarrow{TAS} + \overrightarrow{Windspeed}$$

CWA is the angle between course and wind direction, that can be found by sub-

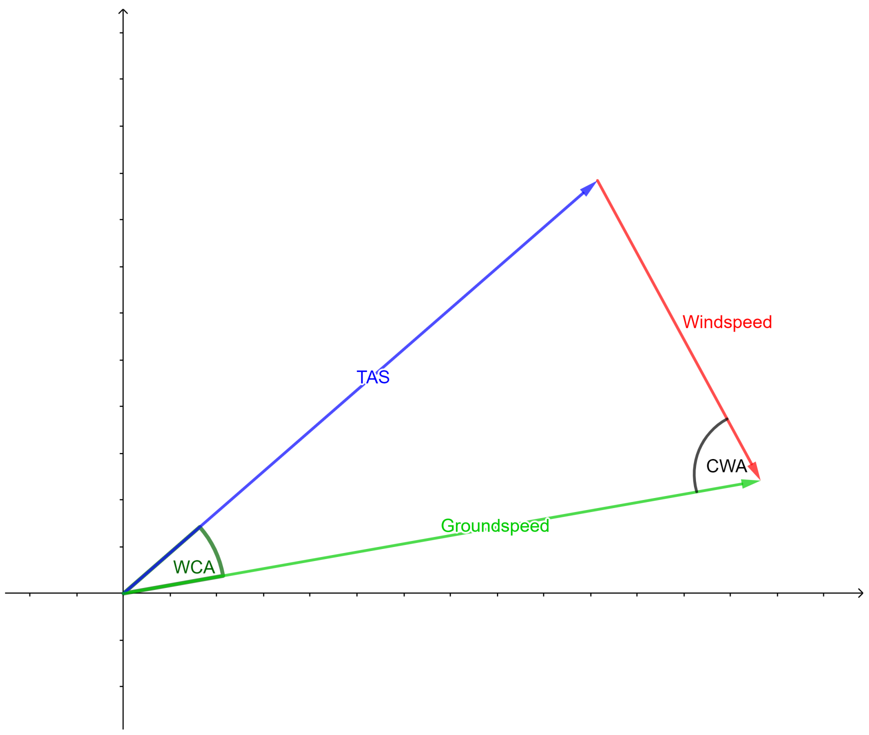


FIGURE 4.3: Wind Triangle

tracting one from another:

$$CWA = Course - Winddirection$$

WCA is the TAS shift in the influence of winds. To find it, we can use the Law of sines:

$$\frac{\sin WCA}{|\overrightarrow{Windspeed}|} = \frac{\sin CWA}{|\overrightarrow{TAS}|}$$

Therefore,

$$WCA = \arcsin\left(\frac{\sin|\overrightarrow{Windspeed}|}{|\overrightarrow{TAS}|\sin CWA}\right)$$

Finally, we can find Groundspeed using the cosine theorem:

$$\begin{aligned}\|Groundspeed\|^2 = \|TAS\|^2 + \|Windspeed\|^2 - \\ - 2\|TAS\| \cdot \|Windspeed\| \cdot \cos\left(180^\circ - WCA - CWA\right).\end{aligned}$$

4.3 Flight Phases

The flight in the environment that we consider is divided into three phases: climb, cruise, and descent.

4.3.1 Climb

The climb phase is started when the airplane reaches an altitude of 10000ft at a constant speed. After that, it accelerates to some TAS that is calculated from IAS. In this case, IAS is a parameter that can be optimized. After that, it reaches crossover altitude where TAS given from IAS and from Mach number are equal. After that TAS is constant and calculated from Mach number up to the point that indicates the end of this phase called TOC.

4.3.2 Descent

The point of beginning the descent is called top of descent. And the logic of this is the opposite to climb. First airplane descents to crossover altitude at constant Mach number, then it accelerates from some IAS to the certain speed that was on the beginning of climb.

4.3.3 Cruise

Cruise is usually the longest flight phase, and it is the main part that should be optimized. This phase begins in the top of climb and end at the top of descent, and it is divided into some waypoints, each of which is a position expressed by latitude and longitude (Figure 4.4). As we can see, there are five possible routes for aircraft, and the red line is the one possible trajectory of aircraft flight. The trajectory is one of the parameters that should be optimized. It seems the trajectory built on only middle points is the most optimal, but due to the influence of winds and weather conditions, it is not always true. Also, there are two more parameters that can be optimized: the altitude where the aircraft is located and Mach number.

4.4 Routes building

The first step of building routes is to construct waypoints over the geodesic line. A geodesic line – is the shortest path between two points on Earth, and then we build a constant number of waypoints that lies on these lines and have the same distance between them. Then we build n almost parallel trajectories from each side. The waypoints of alternative trajectories can be found by doing the following: We take the point a_t from the waypoints of the geodesic line. Find bearing between this point and the last waypoint (TOD). The absolute difference between found bearing and the bearing between a_t and a_t' should be 90 degrees, so we add to that bearing b 90 and 270 degrees b' to find waypoints from both sides. We can calculate latitude and longitude given old waypoint, bearing, and angle distance d between waypoints by the following formula:

$$a_t = (Lat, Lon),\ a_t' = (Lat', Lon')$$

$$Lat' = \arcsin(\sin(Lat)\cos(d) + \cos(Lat)\sin(d))\cos(b')$$

$$Lon' = Lon + \arctan 2(\sin(b')\sin(d)\cos(Lat), \cos(d) - \sin(Lat)\sin(Lat'))$$

Distance is a certain constant number that allows to calculate two new waypoints from both sides. In the figure 4.4 the red trajectory is the geodesic line, and the other four are generated alternative trajectories.

4.5 Aircraft model

As it was written before, in order to build aircraft model we need to use something like PDB. For this purposes we used OpenAP. It provides us with all the necessary

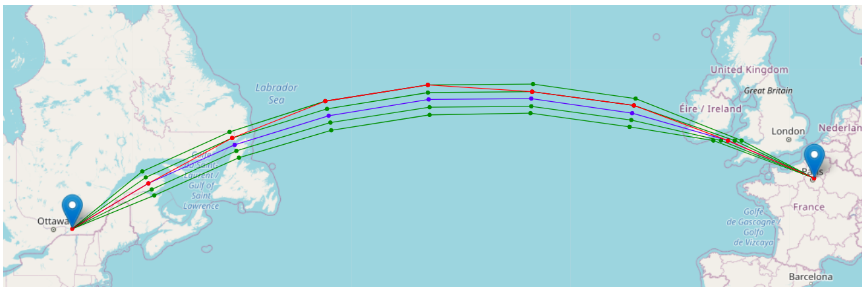


FIGURE 4.4: Alternative Trajectories

information that we can use to build an aircraft performance model for our environment. The main things that we should extract from it are speed, altitude range, and fuel flow. In our case, fuel flow is the amount of fuel burnt in a period of time, and it is measured in kilograms per second, and it depends on TAS, current altitude, and airplane weight. In this work, we set airplane weight to a constant number, so we get the following function:

$$FF(TAS, altitude)$$

In Figure 4.5 one can see how the fuel flow of Airbus210 at a constant weight of

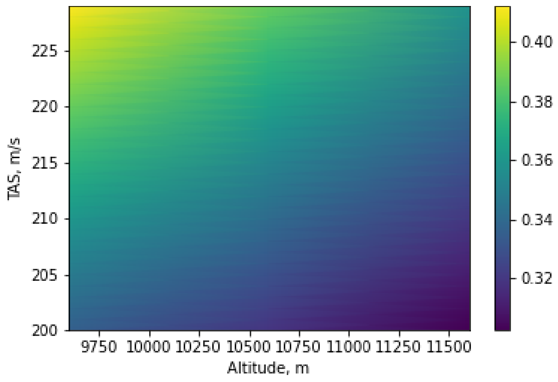


FIGURE 4.5: Fuel Flow kg/s

30000kg depends on TAS and altitude. To decrease fuel flow, we should decrease TAS and increase altitude. So, the optimal solution is being on the highest altitude and with the lowest TAS, but it doesn't always minimize the fuel consumption because the highest altitude is not always available due to weather conditions. Also, TAS depends on winds and it is possible that winds at the lowest and the highest altitude have big magnitude, but at the lowest one its direction coincides with the aircraft course, and at the highest one - on the contrary. That is why it can be better for aircraft to choose the lowest altitude.

4.6 All in one place

To build an environment, we need to define three things: state space, action space, and reward function. We consider only the cruise phase of flight because it is the longest one, and the most fuel is burnt during this phase. Also, the parameters that can be optimized on other phases are different from the ones from the cruise, and it can be complicated to build a general algorithm to handle all the phases.

In general case, state is the location of the aircraft, and it is expressed by the current waypoint and altitude. The current waypoint is the latitude and longitude of the aircraft, which means that we have a 3-dimentional state.

The actions that aircraft can do located in a certain state is to choose one of five next waypoints (trajectory), choose Mach number and the altitude step (for example, aircraft can do climb into 1000ft or 2000ft; or descent in the same value, and also altitude can remain unchanged.

The reward that the agent can get should be related to fuel consumption. The cumulative reward should display fuel consumption during flight. First, we need to find TAS, then we can find groundspeed using TAS and winds direction and magnitude in the current waypoint. The next step is to calculate fuel. Finally, we find the distance between the current waypoint and the previous and calculate time for it using groundspeed. The reward is fuel flow multiplied by travelling time from one waypoint to another. We add some positive constant to its value to avoid negative values, because some algorithms cannot handle them. Also, we should give small reward if the agent extends beyond the altitude limits and if it arrives to final point at a not exact time.

Chapter 5

RL Agents

5.1 Q Table

Q-Table algorithm is a tabular representation of Q-Learning approach. This algorithm works with discrete action and space, that is why before using it we need to define our state and actions to feed them to Q-Table.

5.1.1 State space

As it was mentioned before our state space consists of waypoint and altitude. We can define our waypoint as a tuple of trajectory index and the point index. Assume that we have five different trajectories and nine waypoints in each. Altitude space is defined within minimum cruise altitude and maximum one. For making it possible to run ass a table, we need to divide this range into n values. For example, if we have the altitude range [10000ft, 40000ft] we can divide it into 40 values with step 1000ft and we will have the values: [10000ft, 11000ft, ..., 40000ft]. In that case, our state-space will contain 40x5x9 = 1800 values.

5.1.2 Action space

The action space of our environment is defined by tuple (Mach Number, Altitude Step, trajectory). We need to descretize our mach number in the same manner that we did with altitude. For example, if we have mach number within range [0.75, 0.8] we can divide it into 5 values with step 0.01. Since we have 5 different trajectories, 5 different altitude steps and 5 values of mach number, the action space will contain 5x5x5 = 125 values.

5.2 Deep Q-Networks

Q-Table requires states to be discrete, but what to do with continuous ones? For such a case we need to use Deep Q-Networks. The main idea of this an algorithm is similar to Q-Table. It is also an off-policy value-based TD algorithm, and updating the Q value has similar logic. But instead of directly calculating it, this algorithm approximates it using ANN:

$$Q^*(S_t, A_t) \approx Q(S_t, A_t; \theta)$$

where θ is parameter (weights) of ANN. The loss function for such ANN training looks the following way:

$$L(\theta) = (y_i - Q(s, a; \theta))^2$$

where

$$y_i = R + \gamma \max_{a'} Q(s', a'; \theta).$$

It makes RL more similar to a supervised learning problem, but in such kinds of problems, we can deal with a full dataset with additional preprocessing and shuffling. Shuffling is one of the main steps because, in such a way, we allow our optimization method to avoid developing overfiting biases; reduce the variance of the training process, speed up convergence, and overall learn a more general representation of the underlying data-generating process [10]. But in RL, full dataset is not fully collected for training. The data (R_t, S_t, A_t, S_{t+1}) is constantly gathered, so the nearby samples can be correlated.

The other problem with such kind of training is that targets are moving with every training step, because we change value function, and it can negatively affect the training process. In supervised learning, we always have fixed targets. The idea of avoiding this problem is to introduce the target network:

$$Q(s, a; \theta')$$

It has the same architecture and is initialized with the same weights as a model that we will optimize (current network). This network will be used in the training process with frozen parameters, and after some training steps, its weights will be updated by copying the weights of the current network. The difference in convergence one can see in the Figure 5.1.

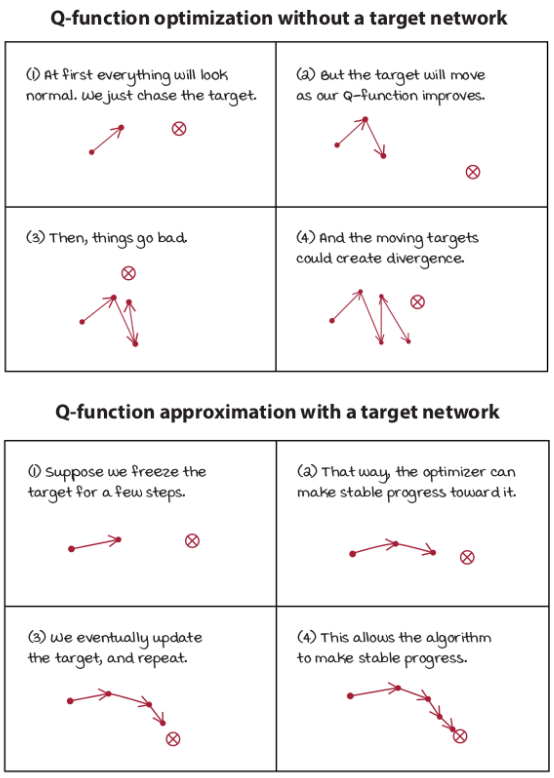

Q-function optimization without a target network
(1) At first everything will look normal. We just chase the target.
(2) But the target will move as our Q-function improves.
(3) Then, things go bad.
(4) And the moving targets could create divergence.
Q-function approximation with a target network
(1) Suppose we freeze the target for a few steps.
(2) That way, the optimizer can make stable progress toward it.
(3) We eventually update the target, and repeat.
(4) This allows the algorithm to make stable progress.

FIGURE 5.1: Training Process. This was taken from [10]

The step of updating the target network can vary due to the problem, and it is a hyperparameter that must be tuned.

5.2.1 Experience Replay

The one possible solution of solving the correlation between data samples is to build an experience replay buffer. It is some data structure that stores played samples in memory. It also has limited memory. For example, it can hold only 1000 or 20000 samples. This buffer can gather data:

$$(R_2, S_1, A_1, S_2), (R_3, S_2, A_2, S_3), \dots, (R_{t+1}, S_t, A_t, S_{t+1})$$

and it allows to sample random mini-batches for model to train. The method reduces the correlation between samples, and the target can be moved slower than better convergence. The overall algorithm with experience buffer is illustrated on 5.2

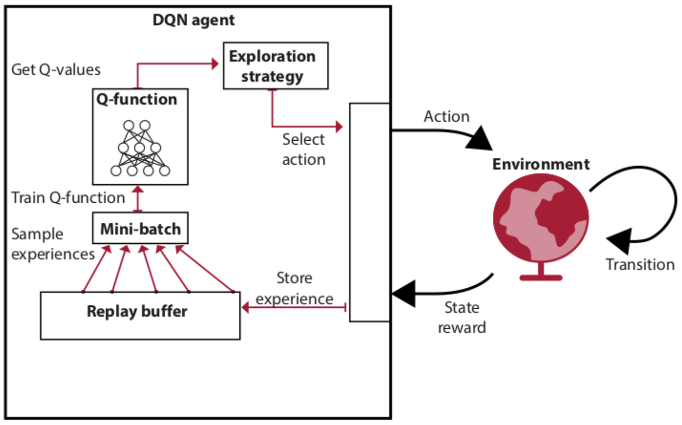


FIGURE 5.2: Training Process. This was taken from [10]

5.2.2 Double Deep Q-Networks

There is a paper [4] where the author has shown that Q-values will cause positively biased results of $\max_a Q(s', a; \theta)$ if $Q(s', a'; \theta)$ contains any errors. There are many reasons of such errors: function approximation using neural networks is not perfect, an agent may not fully explore the environment, and the environment itself may be noisy. [9] So, we can make an improvement of it by following updating rule:

$$Q_{target}(s_t, a_t) = r + \gamma Q(s_{t+1}, \max_a Q(s_{t+1}, a))$$

5.2.3 Dueling DQN

Another improvement of the DQN algorithm is Dueling DQN. This approach affects only ANN architecture. The algorithm and Q-values updating are the same as in DQN. The idea of such approach is to connect Q-function to value function $V(s)$ and action-advantage function $A(s, a)$ by summing it:

$$Q(s, a) = V(s) + A(s, a)$$

In the case of learning only Q-value, we learn different values for different state-action pairs: $Q(s_1, a_1), Q(s_1, a_2), ..., Q(s_1, a_t)$ etc. But as you can see in that sequence, we have the same state s_1, so the idea of such an approach is that instead of learning only the function that depends on state-action, we also learn the value function that is common to all actions $V(s)$.

The idea of Dueling DQN architecture is to use shared and separated layers for value and advantage. For example, if the state is an image, we should use CNN, we can make the convolutional layer shared and build two different fully-connected layers for both estimators: value and advantage.

The motivation of the dueling architecture is to create a new network that improves on the previous network but without having to change the underlying control method. [10]

Finally, Q-value can be computed by following formula:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)$$

Where θ stands for weights shared layer, α and β stand for advantage and value layers respectively. For stabilisation of learning process, we shift our Q-value in a way of subtracting mean of advantage of all actions.

5.2.4 Invalid actions

One of the main problems of training DQN and its improvements is invalid actions. It is the actions that we shouldn't do at all. In our case, we can call the action invalid if the aircraft goes beyond the allowed altitudes. It is what we want to forbid our agent from doing it. There are some possible approaches to avoiding such actions.

The first one is to give the agent a very small reward for these actions. It is acceptable to approach methods when we don't use function approximation. But there is a problem with this approach when we work with DRL. As [7] states, it has a hard time training the forbidden actions. The agent will require a lot of transitions to set small Q values to invalid actions, and it will take time instead to explore the valid, valuable samples, which are very useful for the agent to perform more optimally.

The second approach is not sampling from the forbidden actions and setting the Q-values of such ones to small values. This way can be acceptable in many cases, but there are also some problems with it. First, we need to check all actions every time we are sampling to avoid forbidden ones, and this can take time. The second problem is that we are trying to change the output of value functions manually, and the agent cannot understand and learn which actions are forbidden and which are not.

The third approach is introducing Frontier Loss. This way of solving DQN problem with forbidden actions was suggested in the paper [7]. The authors add a feedback function F to transition tuple - $(S_t, A_t, R_{t+1}, S_{t+1}, F)$. F is a function that takes state and action as arguments and returns 0 if the action is valid and 1 otherwise. The loss function is based on the assumption that an agent can never make illegal decisions following the optimal policy. The rule is the following: for every state encountered during training, the Q-values of all forbidden actions should be below one of each valid action, within a certain margin m. Margin is the hyperparameter that should be chosen according to the reward function.[7] The loss that describes this rule is following:

$$J_F(\theta) = Q(s, a_{invalid}) - \min_{a_{valid}}(Q(s,a) - m)$$

And it is called Frontier Loss. The algorithm that includes such loss is called DQN-F and train networks due to following loss function:

$$J(Q) = \lambda_1 J_{DQN}(Q) + \lambda_2 J_F(Q)$$

Where J_{DQN} is the standard DQN loss and λ_1 and λ_2 are the weights for the losses and are hyperparameters. The Frontier Loss training process can be seen in the Figure 5.3. Such an approach directly puts Q-values of forbidden actions below valid

```
Algorithm 1: Frontier loss and classification network.
  Data: minibatch $b$ from replay buffer $\mathcal{R}$, Q-network $\mathcal{Q}$,
        classification network $\mathcal{C}$
  Result: Frontier loss
  loss = 0;
  for (state $s$, action $a$, feedback $f$) in minibatch $b$ do
      if $f$ = 1 then
          $a^-$ ← a                                  ▷ Renaming for clarity
          actions$_{valid}$ ← C(s);                  ▷ Estimated valid action set
          if min[$Q(s, actions_{valid})$] < $Q(s, a^-)$ - $m$ then
              loss = loss + || min[Q(s, actions$_{valid}$)]
              - Q(s, $a^-$) -m ||$^2$
  return loss;
```

FIGURE 5.3: DQN-*F* Frontier loss training process (taken from [7])

ones.

5.3 Policy Gradient algorithms

This section in mostly based on [9]. The previous methods were based on value functions, and policies were built using them. The policy gradient methods use parameterized policy, and the action is selected by using it, that can be written in the following way:

$$\pi(a|s,\theta) = P(A_t = a|S_t = s, \theta)$$

That means that such policy describes the probability of doing some action, being in a certain state. The goal of RL algorithm is to maximize cumulative reward. We need to train such policy that can achieve this. Objective function can be defined as:

$$J(\pi_\theta) = \mathbf{E}_{\tau\sim\pi_\theta}(\sum_{t=0}^{N} \gamma^t r_t)$$

and then we define policy gradient rule:

$$\max_\theta J(\pi_\theta) = \mathbf{E}_{\tau\sim\pi_\theta}(\sum_{t=0}^{N} \gamma^t r_t)$$

These methods are concentrated of maximizing agent performance. The updating rule looks like gradient ascent:

$$\theta_{t+1} = \theta t + \alpha \nabla J(\theta_t)$$

where $\nabla J(\theta_t)$ is the policy gradient, and it is defined:

$$\nabla_\theta J(\theta_t) = \mathbf{E}_{\tau\sim\pi_\theta}(\sum_{t=0}^{N} \gamma^t r_t \nabla_\theta \log \pi_\theta(a_t|s_t))$$

We can define policy function with respect to weights in any way. In most cases, the policy function uses soft-max distribution so that the actions with the highest values can have higher probabilities and the actions with the lowest - on the contrary. Gradient ascent looks like:

$$\pi(a|s,\theta) = \frac{\exp^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}$$

This allows the agent to perform an action with a certain probability with respect to policy. One of the advantages of policy gradient methods over value-based approach is that we are training the model using a gradient with respect to stochastic policy nature, which means better convergence since the action is changed smoothly, unlike when we are taking a maximum of value functions. Another advantage is despite the fact that the policy is stochastic, we can easily make it deterministic.

5.3.1 REINFORCE

REINFORCE is one of the policy gradient RL algorithms and it considers to be the fundamental one. As I have mentioned before the policy gradient methods use gradient ascent for weights updating, but REINFORCE express loss function as gradient descent:

$$\theta_{t+1} = \theta t + \alpha A(s,a) \nabla_\theta \log \pi(a|s;\theta)$$

5.4 Actor Critic

Actor-Critic algorithms combine policy gradient and value function. Such approaches contain two components: Actor and Critic. Actor is doing actions (it is expressed as parameterized policy), and Critic estimates them (that's how value function works). These components are learned together. The idea of such an algorithm is that the reward value can give less information than the value function. The policy gradient of Actor is quite similar to one from REINFORCE:

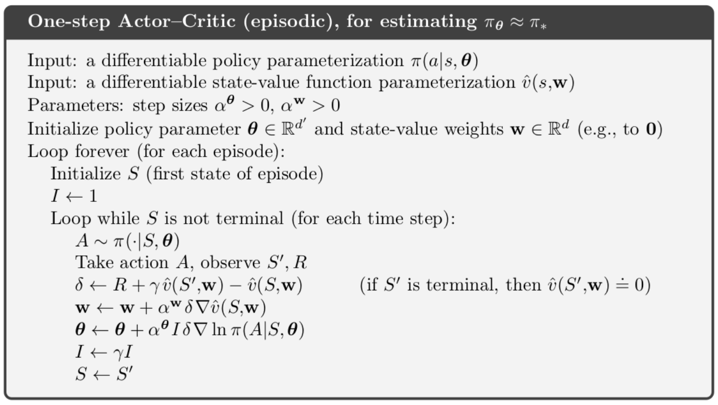
$$\nabla_\theta J(\theta_t) = \mathbf{E}_{\tau \sim \pi_\theta}(A_t^\pi \nabla_\theta \log \pi_\theta(a_t|s_t))$$

where $A(s,a)$ is an advantage function that was used in Dueling DQN. It can be found by subtracting $V(s)$ from $Q(s,a)$:

$$A(s,a) = Q(s,a) - V(s)$$

And the Critic is used for finding $A(s,a)$ The general algorithm of Actor-Critic is defined by the following way 5.4

As we can see from the algorithm, this approach is collecting data before training, that allows making parallel environments for this. See Figure 5.5



```
One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

FIGURE 5.4: Actor Critic algorithm. This was taken from [12]

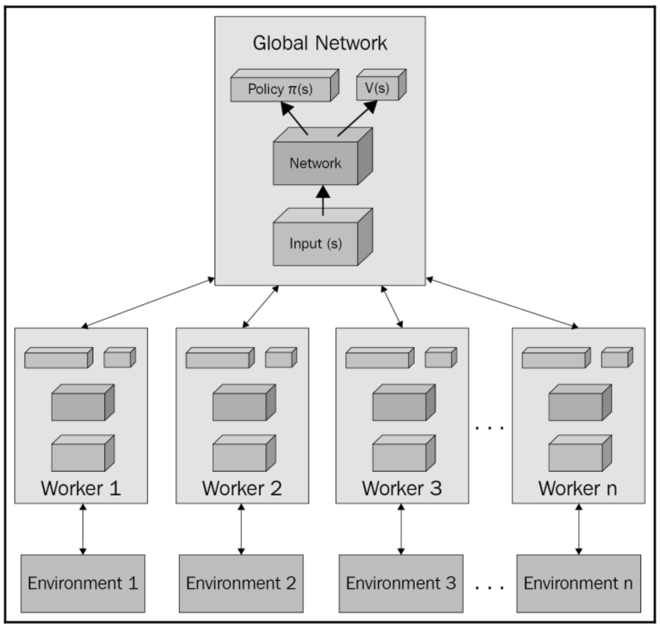


FIGURE 5.5: A2C distributed environments. This was taken from [8]

Chapter 6

Results

The environment and all described algorithms were implemented in Python programming language use PyTorch - framework for dealing with ANN. The route from which it was trained was Montreal to Toronto. The route was divided into 9 way-points, and it has 5 alternative trajectories. The idea of choosing such parameters in environment building is because the [1] uses the same ones, so it was easier to compare with it. As it was said, such kinds of problems are generally solved by genetic algorithms. To find the optimal solution and compare results with RL approaches, we used an open-source scipy optimizer [13] that uses the genetic algorithm [1] It will be also compared with greedy agents.

The main goal is that the agent should perform a solution close to the optimal one from scipy and that it should perform better then greedy agent.

6.1 Training

In this section we will show the results of testing different approaches that were described in previous sections.

6.1.1 Q Table

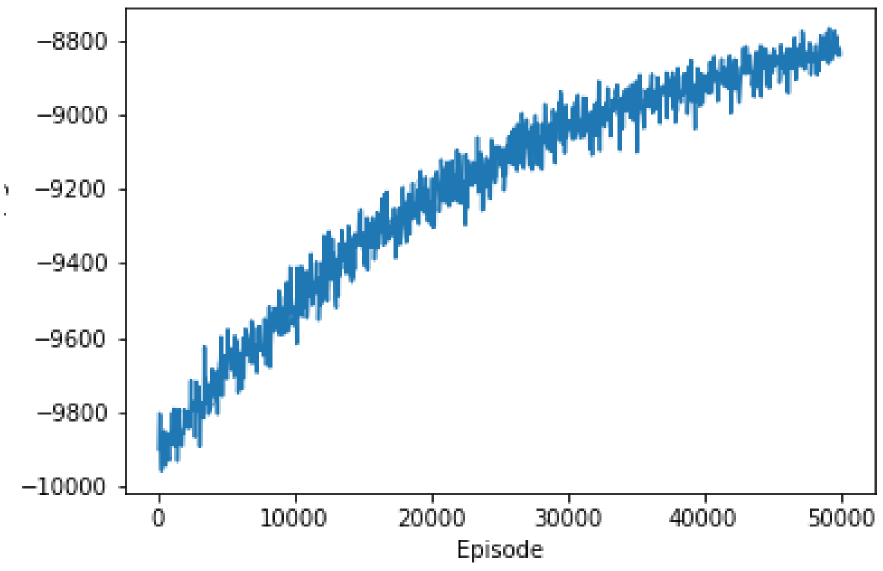


FIGURE 6.1: Fuel consumption comparing

As we can see on Figure 6.1, the learning process of Q-Table is quite successful. On the y-axes, we have the inverse value of episodic fuel consumption. Since reward function is related to fuel consumption by adding a constant to it, the shape of the reward is the same, but the fuel consumption plot was used because it gives more clear values. This approach avoids forbidden action by setting q values to 0. This approach doesn't take a lot of time to train. The agent is training almost perfectly, but there is a problem with this approach that it depends on the dimension of state space since we have a tabular representation of such values. With increasing waypoints or trajectories or discretizing steps, this table will be much larger and take up a lot of space in memory. Also, we cannot complicate the environment by adding weather conditions to the state or something like that in future work. That's why we need DL approaches that handle continuous space.

6.1.2 DQN

Another tested approach is DQN and its improvements. The architecture network of DQN and DDQN has two hidden layers with 128 and 64 neurons with activation function tangent.

Dueling DQN has architecture as value with one hidden layer with 64 neurons and an advantage with hidden layer of 128 and 64 neurons. The activation function is tangent. They are considered in one section since their training process is quite similar. To avoid forbidden actions, we sample from only valid ones when taking random action and set Q-value 0 when we follow policy. We use the target model and update it every 15 steps. The smoothed training process of these approaches can be found in Figure: 6.2. The DQN and DDQN show almost the same performance,

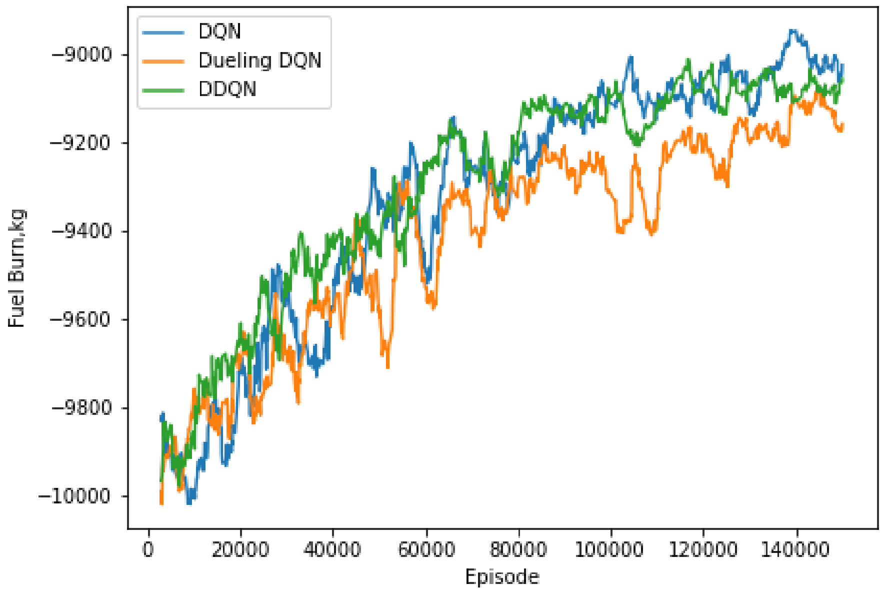


FIGURE 6.2: DQN family Training

but Dueling DQN is worse than them. This performance overall is worse than Q-Table because, in this case, we are using function approximation rather than directly calculating Q-value, that cost more chance for error.

6.1.3 DQN-F

Another approach is called DQN-F. In this case, we let the agent do forbidden actions so that he can learn to avoid them. We set the margin to 100 since we have quite a big reward and the weight of the frontier loss of 0.3 so that it can pay more attention to finding the optimal solution. The smoothed training process is shown in the Figure 6.3. This approach teaches to deal with forbidden actions, but it shows worse

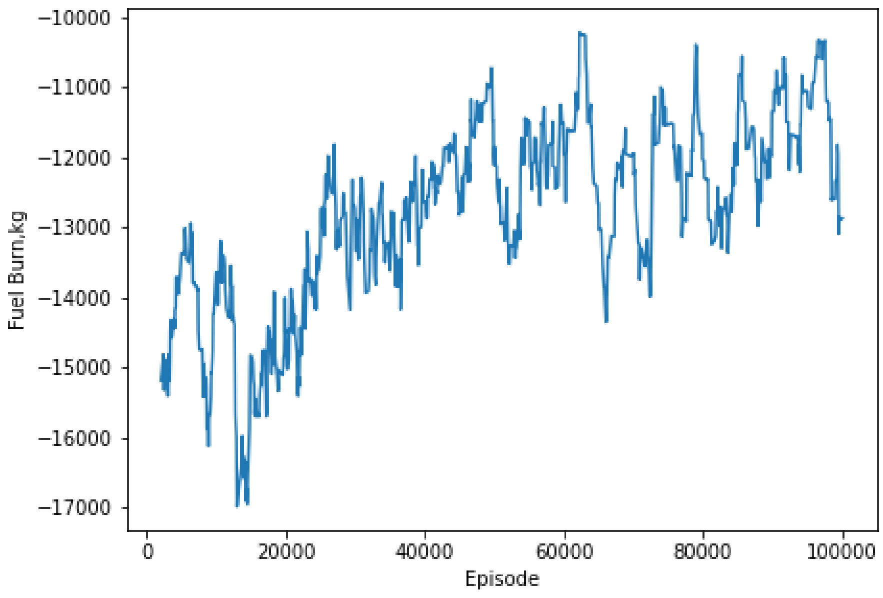


FIGURE 6.3: DQN-F Training

performance in finding the optimal solution. The approach of just not to sampling from invalid actions is better.

6.1.4 A2C

The next agent that was trained was A2C. For its training, we used three numbers of parallel environments and a small learning rate. In the architecture of ANN, only one shared hidden layer was used for both actor and critic with 32 neurons and an activation function tangent.

You can see the process of training on the Figure: 6.4. It is learning pretty well although it requires many episodes due to small learning rate. The performance of this approach is similar to Q-Table.

6.1.5 Compare Results

The next step is to compare how every agent performs in a static flight path environment, including greedy agent and solver.

DQN	DDQN	Dueling DQN	A2C	QTable	DQN-F	Solver	Greedy
8914.1	8788.6	9012.45	8688.71	8618.3	9838.04	8671.04	8829.62

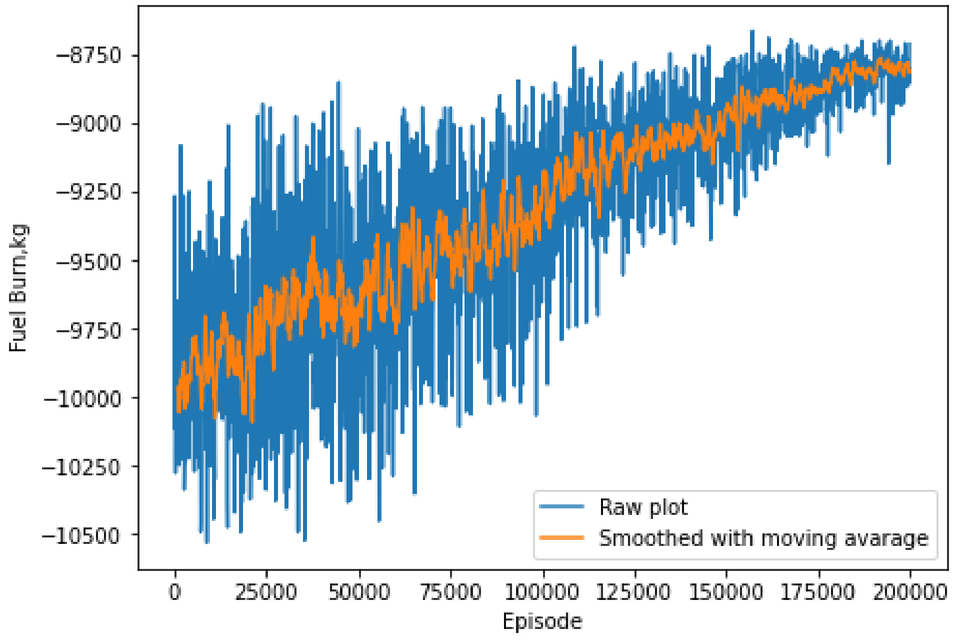


FIGURE 6.4: Fuel consumption comparing

As you can see on the table Q-table showed the best result, and A2C is quite similar to the solver and is better than greedy. DQN family has worse performance in this environment because they are losing to greedy algorithm. If to compare the results of Q-Table (since it showed the best results), Greedy agent, and Solver agent in a static environment, the Mach number in all cases is almost the same. In the case of Q-Table and Greedy agent, its value is 0.75 in all waypoints, but in solver agent, its mean value is 0.75144. The trajectories of the agents you can see on Figures 6.7, 6.5 and 6.6.

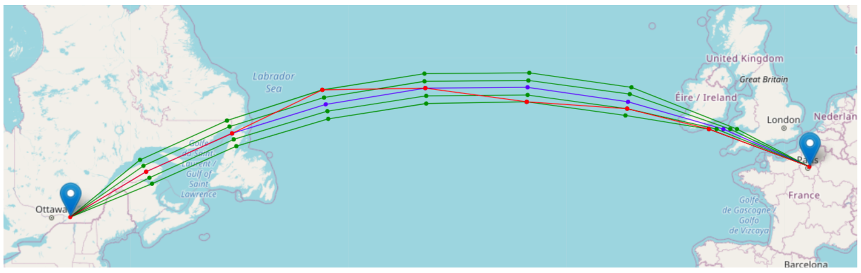


FIGURE 6.5: Greedy trajectory

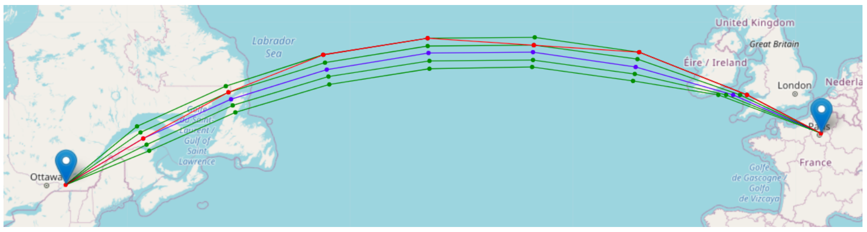


FIGURE 6.6: Solver Trajectory

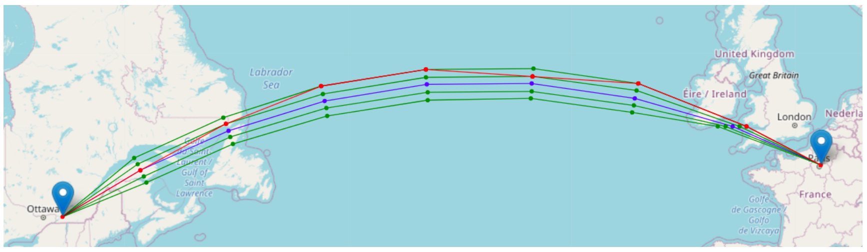


FIGURE 6.7: Q Table trajectory

As can see, the trajectory of the greedy agent differs from the other ones. That means that the routes in the trajectory in which in every step we get less fuel consumption is not optimal, the solver agent and Q-Table are choosing such routes where the cumulative fuel consumption is the lowest. Let's now look at the altitudes in Figure 6.8

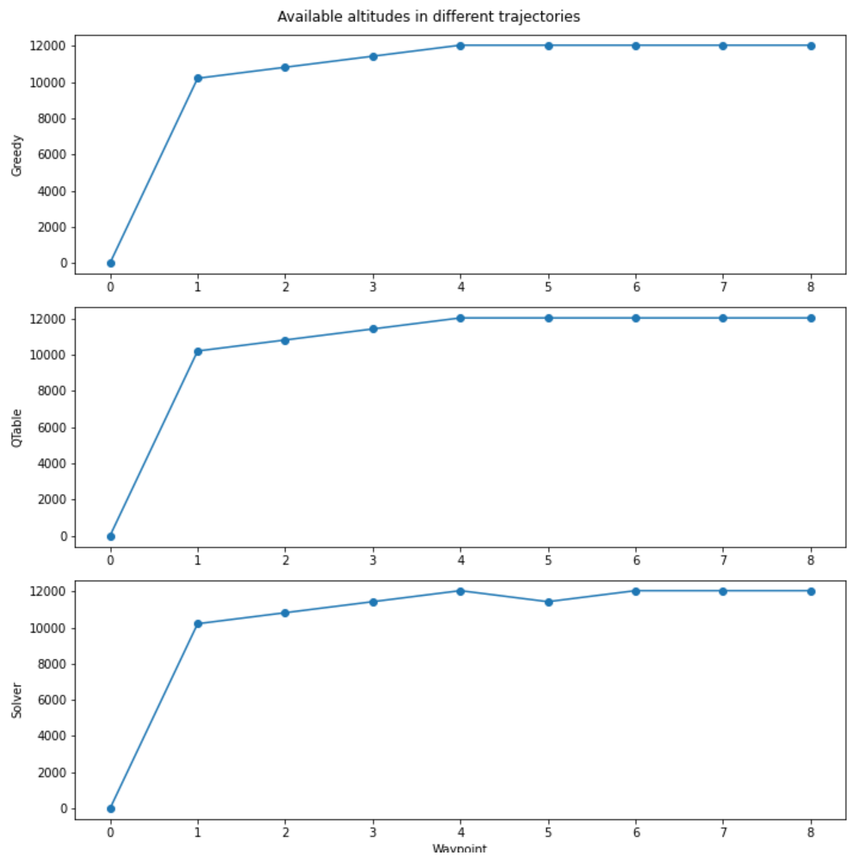


FIGURE 6.8: Greedy trajectory

As it was said before, the lowest fuel flow is at the highest altitude, so the greedy act in this way. In this case, the optimal altitude coincides with the greedy, which means that Q-table also chooses the highest altitude. The solver may not choose the highest one because of the numerical accuracy during calculation. In the case of Mach number, we see that 0.75 seems to be optimal, but the solver has chosen another one, and that could cause the selection of another altitude. That proves that the solution is not perfectly optimal, but it is close to it. Even Q-table shows better results. It doesn't mean that there is no better solution. But it is still close to the optimal one. Anyway, both solver's and q table solutions are close to optimal, and they are better than the greedy algorithm. To conclude, the main idea of the research was proven.

6.2 Dynamic Environment

All the things that have been described before were related to a static environment. That means that we need to know winds' direction, magnitude and weather conditions in advance and then, among all possible routes, find the optimal one. But the initial idea of this project is to describe the environment that the agent can choose paths in dynamic weather. Let's take a look at general weather conditions and how they can affect flight path planning.

The messages that describe such weather conditions are called AIRMET and SIGMET. An AIRMET is a message containing information issued by a meteorological watch office concerning the occurrence or expected occurrence of specified en-route weather phenomena that may affect the safety of low-level aircraft operations and which was not already included in the forecast issued for low-level flights in the flight information region concerned. [14]. SIGMET information is information issued by a meteorological watch office concerning the occurrence or expected occurrence of specified en-route weather phenomena that may affect the safety of aircraft operations [14]. These messages contain information about a thunderstorm, strong winds, icing, turbulence, etc. Such a dangerous zone can be illustrated as polygons. See Figure 6.9

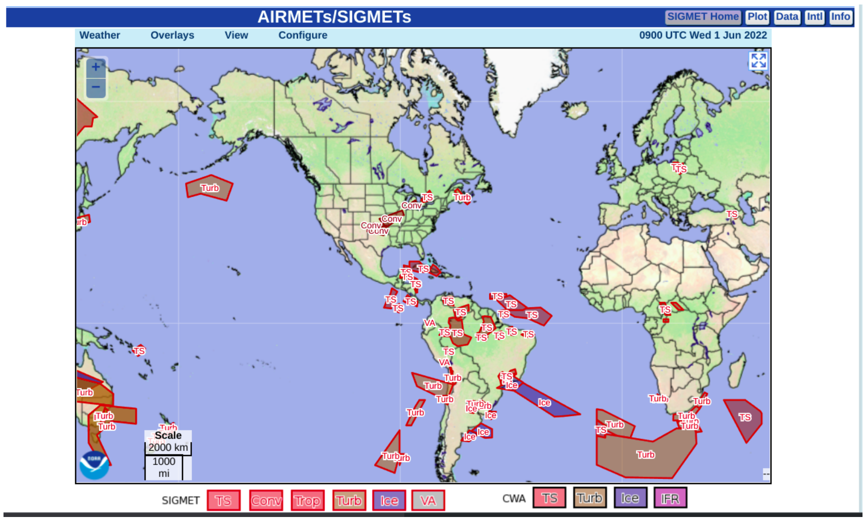


FIGURE 6.9: Weather Conditions. This is taken from [3]

There is also information about their coordinates, speed, direction, and altitude. The information about this thing is available on [3], but as far as it is real-time, and it might be complicated to collect historical data. In our case, we can simulate these zones by limiting altitude in some waypoints. We create a matrix 5x9 (5 trajectories, 9 waypoints) and fill it with maximum cruise altitude. After that, we randomly choose the waypoints and fill them with a value from a uniform distribution of the range between minimal and maximal cruise altitude (Figure 6.10). As we can see in the four of five possible trajectories, and there are some altitudes that agent cannot visit. Let's randomly choose several such environments and check A2C, QTable and Greedy agent on them (see Figure 6.11). In this case, Q-Table and A2C perform much

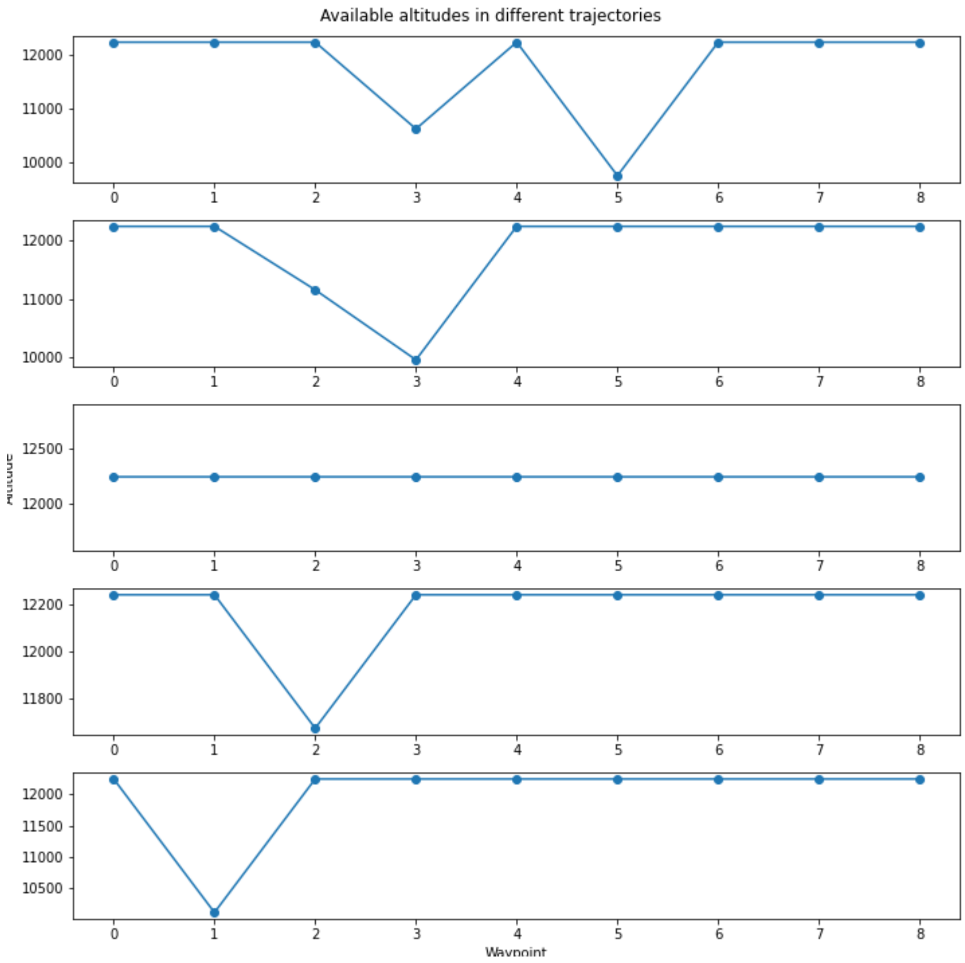


FIGURE 6.10: Available Altitudes

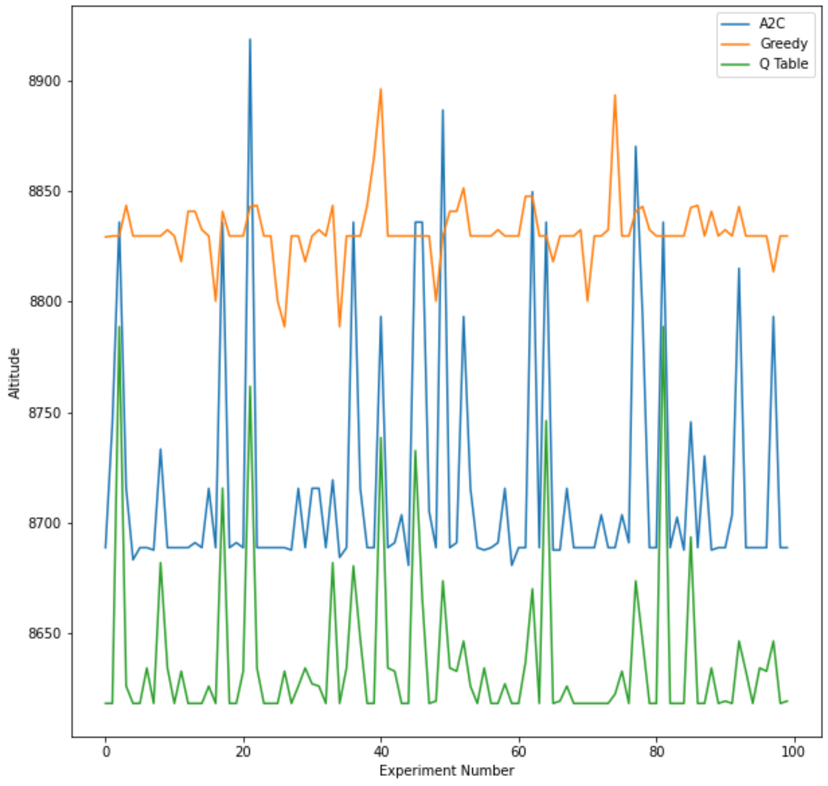


FIGURE 6.11: Comparison of dynamic environment

better than greedy one, that allows to conclude that RL approaches can show good performance in dynamic weather environment.

Chapter 7

Conclusions and Future work

Although, there are many papers and different approaches to solving flight planning problems, this task is still open to new interesting, and nonstandard ideas. As we could see, RL shows quite good results, even compared with the nonlinear solver. Even being trained in a static environment, it can perform quite well in the dynamic one. It is a good step for solving flight planning problems with dynamic weather.

The problem with such approach is that RL agent does not see the overall state of the weather. It just estimates the possible action being in certain states, and the solution in such a case is not optimal. That is why we need DRL even if Q-Table works well. The idea is to represent weather conditions, wind direction, and magnitude with some continuous state. It can be the image where the polygons are represented, and also to add to this state the direction, magnitude, etc. This approach is more complicated and requires approaches that can work with the continuous state like A2C, DQN, etc.

Bibliography

[1] *Aircraft Flight Plan Optimization with Dynamic Weather and Airspace Constraints.* https://www.researchgate.net/publication/271014315_New_methods_of_optimization_of_the_flight_profiles_for_performance_database-modeled_aircraft.

[2] *Aircraft Flight Plan Optimization with Dynamic Weather and Airspace Constraints.* https://core.ac.uk/download/pdf/334963344.pdf.

[3] *Aviation Weather Center.* https://www.aviationweather.gov/sigmet.

[4] *Deep Reinforcement Learning with Double Q-learning.* https://arxiv.org/abs/1509.06461.

[5] *GDPS data in GRIB2 format: 25 km.* https://weather.gc.ca/grib/grib2_glb_25km_e.html#variables.

[6] *Genetic Algorithm – an Approach to Solve Global Optimization Problems.* https://www.ijcse.com/docs/IJCSE10-01-03-29.pdf.

[7] *I'm Sorry Dave, I'm Afraid I Can't Do That Deep Q-Learning from Forbidden Actions.* https://arxiv.org/pdf/1910.02078.pdf.

[8] Micheal Lanham. *Hands-On Reinforcement Learning for Games.* Packt Publishing, 2020.

[9] Wah Loon Keng Laura Graesser. *Foundations of Deep Reinforcement Learning, An Introduction.* Pearson Education, Inc., 2020.

[10] Miguel Morales. *Deep Reinforcement Learning.* Manning Publications Co., 2020.

[11] *OpenAP: An Open-Source Aircraft Performance Model for Air Transportation Studies and Simulations.* https://www.researchgate.net/publication/343163153_OpenAP_An_Open-Source_Aircraft_Performance_Model_for_Air_Transportation_Studies_and_Simulations.

[12] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning. An Introduction.* 2nd ed. aa, 2018.

[13] *Scipy, differential evolution.* https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html.

[14] *Skylibrary.* https://skybrary.aero/articles/airmet.