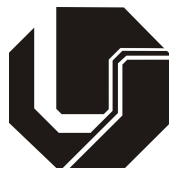


**Rodolfo Augusto Serra Sammarco Branco**

**AI techniques applied to alleviate wind gust in airplanes**



UNIVERSIDADE FEDERAL DE UBERLÂNDIA  
FACULDADE DE ENGENHARIA MECÂNICA

Dezembro 2019

**Rodolfo Augusto Serra Sammarco Branco**

**AI techniques applied to alleviate wind gust in airplanes**

**Projeto de Conclusão de Curso** apresentado ao Curso de Graduação em Engenharia Aeronáutica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de **BACHAREL EM ENGENHARIA AERONÁUTICA**.

Área de concentração: Aeroelasticidade e cargas.

Orientador: Prof. Dr. Tobias Souza Morais

UBERLÂNDIA - MG

Dezembro 2019

BRANCO, R. A. S. S. 2019. **AI techniques applied to alleviate wind gust in airplanes**, Universidade Federal de Uberlândia, Uberlândia.

## **ABSTRACT**

Aircraft under the effect of certain phenomena, for example wind gusts, suffers a modification in the load distribution, this may bring undesirable effects like passengers' discomfort and aircraft structural damages. The use of control surfaces may alleviate the impacts of such phenomena.

Nowadays most of times classical control techniques as PID are used (WRIGHT; COOPER, 2008), nonetheless, it is interesting to apply some of the new artificial intelligence methods to solve real-world problems, the reason being that they may produce better results. So, the objective of the present work is to apply one of these methods, the Reinforcement learning technique, which has the advantage of learning by itself, in a binary flutter system with a control surface in order to minimize the amplitude of the wing's response.

*Keywords: artificial intelligence (AI), reinforcement learning (RL), machine learning (ML), gust control, aeroelasticity, python.*

## LIST OF FIGURES

2.1	Collar triangle, i.e., aeroelasticity triangle. . . . .	3
2.2	Aeroservoelasticity pyramid. . . . .	3
2.3	Binary system based on the one presented in (WRIGHT; COOPER, 2008), chapter 11. . . . .	4
2.4	Illustrating RL. . . . .	7
2.5	Maze: Initial state. . . . .	7
2.6	Maze: shortest path. . . . .	7
2.7	Maze: A path. . . . .	8
2.8	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	9
2.9	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	9
2.10	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	10
2.11	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	10
2.12	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	11
2.13	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	11
2.14	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	12
2.15	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	12
2.16	Maze: Rewards. Where $j$ is the number of moves used by the player. . . . .	14
3.1	Wing tip lead displacement in meters of a network trained with the reward R1, given a result score of 408. . . . .	21
3.2	Surface control position of a network trained with the reward R1, giving a result score of 408. . . . .	21

3.3	Wing tip leading edge displacement in meters of an AI agent trained with the reward R4. . . . .	22
3.4	Surface control position of an AI agent trained with the reward R4. . . . .	23
3.5	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 5038.76. . . . .	24
3.6	Surface control position of an AI agent trained with the reward R6, giving a result score of 5038.76. . . . .	24
4.1	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 47882.96. . . . .	27
4.2	Surface control position of an AI agent trained with the reward R6, giving a result score of 47882.96. . . . .	28
4.3	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 4126.34. . . . .	29
4.4	Surface control position of an AI agent trained with the reward R6, giving a result score of 4126.34. . . . .	29
4.5	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 30901.44, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s. . . . .	30
4.6	Surface control position of an AI agent trained with the reward R6, giving a result score of 30901.44, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s. . . . .	31
4.7	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s. . . . .	31
4.8	Surface control position of an AI agent trained with the reward R6, giving a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s. . . . .	32

4.9	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 4803.39, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s. . . . .	32
4.10	Surface control position of an AI agent trained with the reward R6, giving a result score of 4803.39, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s. . . . .	33
4.11	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and <i>NN2</i> , given a result score of 21895.78, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s. . . . .	34
4.12	Surface control position of an AI agent trained with the reward R6 and <i>NN2</i> , giving a result score of 21895.78, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s. . . . .	34
4.13	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and <i>NN2</i> , given a result score of 8271.52, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s. . . . .	35
4.14	Surface control position of an AI agent trained with the reward R6 and <i>NN2</i> , giving a result score of 8271.52, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s. . . . .	35
4.15	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and <i>NN2</i> , given a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s. . . . .	36
4.16	Surface control position of an AI agent trained with the reward R6 and <i>NN2</i> , giving a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s. . . . .	36
4.17	Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of -92600.48, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s. . . . .	37
4.18	Surface control position of an AI agent trained with the reward R6, giving a result score of -92600.48, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s. . . . .	37

5.1 Wing tip leading edge displacement in meters of an AI agent trained with the re-  
ward R6 and a total of 100 games. Considering the airplane velocity 155m/s. . . . 42

5.2 Control surface position of an AI agent trained with the reward R6 and a total of  
100 games. Considering the airplane velocity 155m/s. . . . . 42

## LIST OF TABLES

2.1	Q-table, where $N$ represents the impossible moves. . . . .	13
2.2	Q-table, where $N$ represents the impossible moves. . . . .	13
2.3	Q-table, where $N$ represents the impossible moves. . . . .	14
4.1	Performance evaluation of the results. . . . .	38
4.2	Performance evaluation of the results $NN1$ vs $NN2$ , highlighted in green the best result given for one velocity. . . . .	39
5.2	Neural Network simplified used for the AI agent, also with 65 games of training. . .	44
5.1	Neural Network used for the successful AI agent with 65 games. . . . .	44
5.3	Neural Network used for flutter condition. . . . .	45



## SUMMARY

<b>LIST OF FIGURES</b>	<b>vii</b>
------------------------	------------

<b>LIST OF TABLES</b>	<b>viii</b>
-----------------------	-------------

<b>1 Introduction</b>	<b>1</b>
1.1 Introduction . . . . .	1
<b>2 Bibliographical Review</b>	<b>2</b>
2.1 Aeroservoelastic Problem . . . . .	3
2.1.1 Into the subject . . . . .	3
2.1.2 Binary Flutter System with a Control Surface . . . . .	3
2.2 Reinforcement learning . . . . .	5
2.2.1 Into the subject . . . . .	5
2.2.2 Definitions . . . . .	6
2.2.3 Toy example . . . . .	6
2.2.4 Q-Learning . . . . .	9
2.2.5 RL Dilemmas . . . . .	12
2.2.5.1 Exploration x Exploitation . . . . .	12
2.2.5.2 Defining rewards . . . . .	13
2.2.6 Deep Q-Learning . . . . .	14
<b>3 Methodology</b>	<b>16</b>
3.1 Case Study Implementation . . . . .	17
3.2 Numerical Simulations . . . . .	20
3.2.1 Rewards . . . . .	20

3.2.1.1	R1 . . . . .	20
3.2.1.2	R4 . . . . .	22
3.2.1.3	R6 . . . . .	23
3.2.2	Neural Networks . . . . .	25
<b>4</b>	<b>Results</b>	<b>26</b>
4.1	Results analysis and evaluation . . . . .	27
4.1.1	Analysis . . . . .	27
4.1.1.1	NN1 vs NN2 . . . . .	33
4.1.2	Evaluation . . . . .	38
4.1.2.1	NN1 vs NN2 . . . . .	39
<b>5</b>	<b>Conclusions</b>	<b>40</b>
5.1	Conclusion . . . . .	40
5.2	Future work . . . . .	41
	<b>Bibliography</b>	<b>43</b>
5.3	Appendix A - Neural Networks used . . . . .	44

# CHAPTER I

## Introduction

### 1.1 Introduction

In this work it will be presented an Artificial Intelligence technique known as Reinforcement Learning, applied in a new context, which is the gust-load alleviation in airplanes. The gust-load alleviation beyond increasing the passenger's cabin comfort, increases the performance of the aircraft consumption given that the gust increases the total lift of the aircraft. Since the induced drag is directly related to the lift, if the induced drag is reduced the required thrust is also reduced, as well, the consumption.

Firstly, it will be explained and detailed the aeroelastic problem and the reinforcement learning, its pros and cons for solving the problem. Secondly, the code that would be used itself was unraveled using pseudo-codes and a link<sup>1</sup> to access all codes in python is given. Thirdly, some key points, the network structure and the rewards used in our study case will be shown. Later the results are analyzed qualitatively and quantitatively (using 3 criteria). Finally, the conclusions and the future work from the authors point of view.

---

<sup>1</sup>Link to the codes: <https://github.com/rassbr/AI-techniques-applied-to-alleviate-wind-gust-in-airplanes>

---

## **CHAPTER II**

### **Bibliographical Review**

## 2.1 Aeroservoelastic Problem

### 2.1.1 Into the subject

In (DOWELL et al., 2015), aeroelasticity is defined as the "physical phenomena which involve significant mutual interaction among inertial, elastic and aerodynamic forces". This defines the Collar diagram, also known as aeroelastic triangle in Figure 2.1.

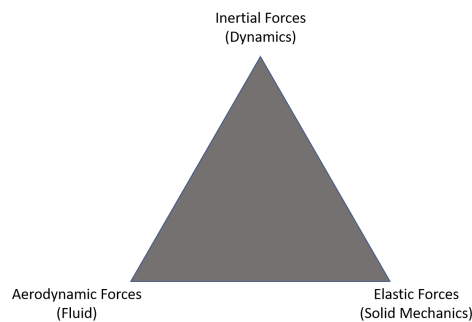


Figure 2.1: Collar triangle, i.e., aeroelasticity triangle.

Aeroservoelasticity extends Collar's diagram into a pyramid including control forces, as defined in (WRIGHT; COOPER, 2008). The aeroservoelasticity pyramid is represented in Figure 2.2, and it illustrates that this time the phenomena involve mutual interaction among inertial, elastic, aerodynamic and control forces.

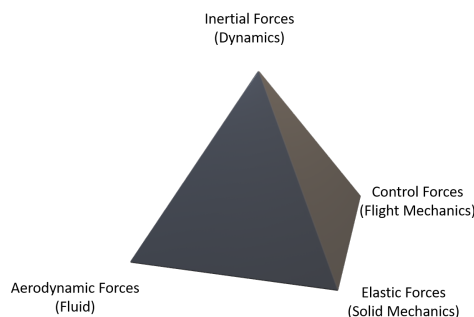


Figure 2.2: Aeroservoelasticity pyramid.

### 2.1.2 Binary Flutter System with a Control Surface

Our problem is based on the binary flutter system with a control surface which is presented in the chapter 11 of (WRIGHT; COOPER, 2008), the Figure 2.3 illustrates it.

To make the lecture easier we copy here the equations 11.1 and 11.2 from (WRIGHT; COOPER, 2008) in the equations (2.1) and (2.2). In the next equation we have the lift and pitching

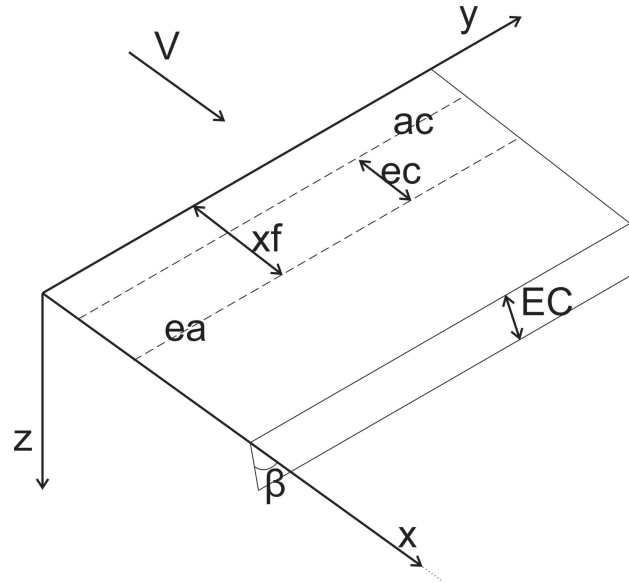


Figure 2.3: Binary system based on the one presented in (WRIGHT; COOPER, 2008), chapter 11.

moment that acts in an elemental strip of the wing.

$$\begin{aligned} dL &= \frac{1}{2} \rho V^2 c dy \left[ a_w \left( \frac{y^2}{s^2 V} \dot{q}_b + \frac{y}{s} \dot{q}_t \right) + a_c \beta \right] \\ dM &= \frac{1}{2} \rho V^2 c^2 dy \left[ e a_w \left( \frac{y^2}{s^2 V} \dot{q}_b + \frac{y}{s} \dot{q}_t \right) + M_{\dot{\theta}} \frac{cy}{4sV} \dot{q}_t + b_c \beta \right] \end{aligned} \quad (2.1)$$

where  $\rho$  is the air density,  $V$  is the aircraft speed,  $c$  is the chord,  $dy$  represents the elemental strip size,  $a_w$  is the lift curve slope,  $y$  the position in the axes of same name,  $s$  is the wingspan,  $q_t$  and  $q_b$  are the generalized coordinates,  $a_c$  is the aerodynamic center position,  $\beta$  is the angle between formed by the wing and the control surface,  $e$  is eccentricity between the elastic axis and the aerodynamic center at 1/4 chord,  $M_{\dot{\theta}}$  is a term to allow simple unsteady aerodynamic effects on the airfoil.

$$\begin{aligned} & \begin{bmatrix} A_{bb} & A_{bt} \\ A_{tb} & A_{tt} \end{bmatrix} \begin{Bmatrix} \ddot{q}_b \\ \ddot{q}_t \end{Bmatrix} + \rho V \begin{bmatrix} \frac{ca_ws}{10} & 0 \\ -\frac{c^2 ea_ws}{8} & -\frac{c^3 M_{\dot{\theta}} s}{24} \end{bmatrix} \begin{Bmatrix} \dot{q}_b \\ \dot{q}_t \end{Bmatrix} \\ & + \left( \rho V^2 \begin{bmatrix} 0 & \frac{ca_ws}{8} \\ 0 & -\frac{c^2 ea_ws}{6} \end{bmatrix} + \begin{bmatrix} \frac{4EI}{s^2} & 0 \\ 0 & \frac{GJ}{s} \end{bmatrix} \right) \begin{Bmatrix} q_b \\ q_t \end{Bmatrix} = \rho V^2 \begin{Bmatrix} -\frac{ca_cs}{6} \\ \frac{c^2 b_cs}{4} \end{Bmatrix} \beta \end{aligned} \quad (2.2)$$

where  $A_{bb}$ ,  $A_{tt}$  and  $A_{bt} = A_{tb}$  are respectively the inertia matrix taking in account the bending, torsion and both directions,  $EI$  and  $GJ$  are the bending and torsion stiffness respectively. This

equation can be rewritten for convenience in the following form:

$$A\ddot{q} + \rho V B \dot{q} + (\rho V^2 C + E)q = g\beta \quad (2.3)$$

where  $g$  represents the influence of the control surface.

We should also include the gust effect, which is an upward wind in the z-direction with velocity  $w_g$ . In (WRIGHT; COOPER, 2008) equation (11.5) we have the lift and pitching moments acting on an elemental strip of the wing due to gust, which is here rewritten in the Equation (2.4):

$$\begin{aligned} dL &= \frac{1}{2} \rho V^2 c dy a_w \frac{w_g}{V} \\ dM &= \frac{1}{2} \rho V^2 c^2 dy e a_w \frac{w_g}{V} \end{aligned} \quad (2.4)$$

which will result in an additional term to our Equation (2.3):

$$h w_g = \rho V \left\{ \begin{array}{c} -\frac{c a_w s}{6} \\ \frac{c^2 e a_w s}{4} \end{array} \right\} w_g, \quad (2.5)$$

thus the system can be rewritten as:

$$A\ddot{q} + \rho V B \dot{q} + (\rho V^2 C + E)q = g\beta + h w_g \quad (2.6)$$

## 2.2 Reinforcement learning

### 2.2.1 Into the subject

In this work we merely intend to give a brief description concerning reinforcement learning, its principles and applications. The reason for this is trying to maintain a concise work. Please look into (SUTTON; BARTO, 2018), which gives a complete and somehow detailed overview into the subject.

Reinforcement Learning (*RL*) is an area of the machine learning. As stated in (Kaelbling; Littman; Moore, 1996), it is where an agent learns a behavior through trial-and-error interactions with a dynamic environment. We could think of RL as a person who never saw a video game trying to learn how to play a game. You give him the controller and tell him where his score is displayed and let him play. The person will start doing random things to see how his actions affect the score, and when he discovers the actions which give him more points, he will start to repeat them, while trying some new movements to maximize the score. Finally, it will always use the same strategy which gives the maximum score that he obtained.

Differently to other machine learning methods where one requires labeled data so the Artificial intelligence (*AI*) can learn, as seen in the analogy before, in RL the AI is guided by a reward. Or, in other words, it is a "goal-seeking agent" (Sutton; Barto, 2018).

### 2.2.2 Definitions

We will define the main terms used in this work of RL taking as reference the definitions given in (Gollapudi, 2016).

- Agent: the entity that is the learner and decision maker;
- Environment: the entity that produces the new situation given an action of the agent, it is also responsible for giving the reward to the agent;
- Reward: it is a numerical feedback from the action taken by the agent, usually a short-term benefit;
- State: is the situation itself;
- Policy: definition of how the agent will behave, we could say, the criteria that guides the actions of an agent.

### 2.2.3 Toy example

To make it easier to understand we will use the Figure 2.4 and take an example, a maze, shown in Figure 2.5. The schema illustrates how the agent using a policy takes an action, this action is given to the environment, which can be a black box, i.e., we may not know how it works



or the criteria to give a reward. The environment returns the state and reward resulting from the previous action to the agent, and it goes on like this.

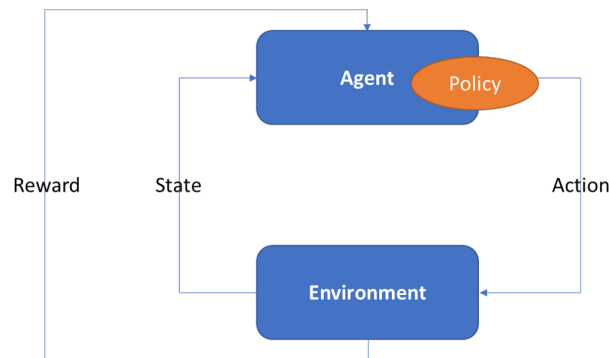


Figure 2.4: Illustrating RL.

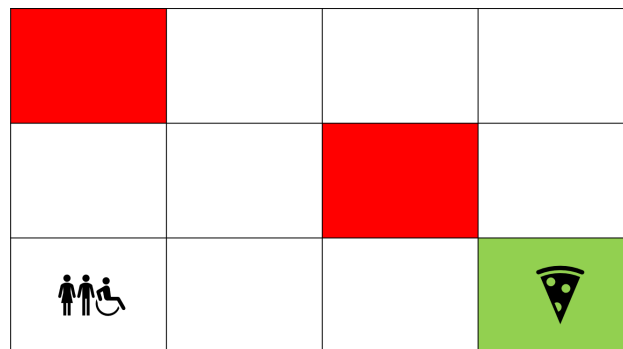


Figure 2.5: Maze: Initial state.

The objective of the group in Figure 2.5 is to arrive until the green square where the pizza is, avoiding the fire, which is represented by the red squares. It is really simple, and in the Figure 2.5 we have our initial state. As humans we now that the shortest path is the one represented in Figure 2.6.

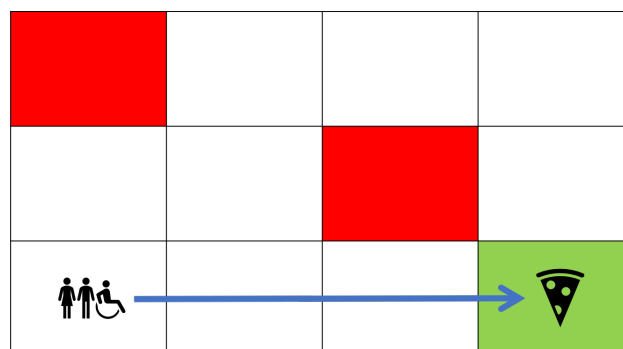


Figure 2.6: Maze: shortest path.

However this is not the only path to arrive until the pizza, we could imagine the one described in Figure 2.7.

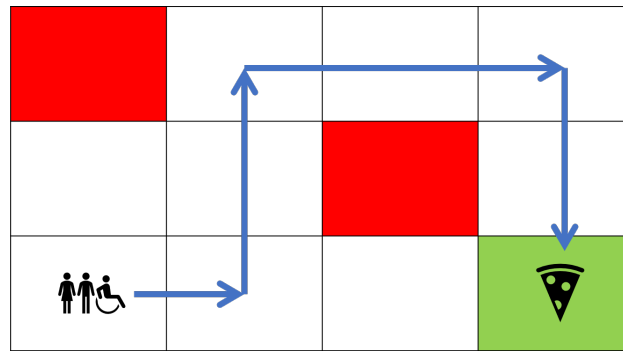


Figure 2.7: Maze: A path.

The best agent should be able to go from the start point until the pizza using the minimum number of possible moves. A regular agent should be able to go from the start point and get to the pizza, without caring about the number of moves. Now that we defined our objectives, we should set a reward for the agent in the grid and make some rules. The rules are:

- If the player goes into the red zone, it is game-over;
- The player can do 15 moves before an automatic game-over;
- The player can only move in the vertical or horizontal direction and one square at time;
- Each time he goes in a square, he is given the reward of that square.

– And the reward that each square gives is defined in the Figure 2.8.

As we can see every white square gives the player a total of 1 point, except for the starting square which gives -10 points, in order to avoid returning to this point, the red squares are game-over and gives a total of -50 points and the green square 15 points minus two times the number of moves used by the player. If we think like this, the shortest path has 3 moves and will give a total of 11 points, the path described in Figure 2.7 has seven moves and will give a total of 7 points. Making these calculations, it seems that we have a reward that gives a maximum of points to the best solution.

Just to make it clear, we are going to make a few moves, the first one is going to the right, this would gives us 1 point and we would be in the position showed in Figure 2.9.

Next we can go again to the right, receive another point and be in the position A shown in Figure 2.10 or go up, receive one point and be in the position B shown in Figure 2.11.

-50	1	1	1
1	1	-50	1
-10	1	1	$15-2*j$

Figure 2.8: Maze: Rewards. Where  $j$  is the number of moves used by the player.



			

Figure 2.9: Maze: Rewards. Where  $j$  is the number of moves used by the player.

From position A we can go to the right, Figure 2.13, and end the game arriving at the pizza receiving 9 points summing up a total of  $1 + 1 + 9 = 11$  points, from position B we can also go right, Figure 2.12 and receive -50 points and a game-over summing up a total of  $1 + 1 - 50 = -48$  points. These are only two examples of how to play. We could also have a game-over later playing as shown in Figure 2.14.

Each position the player passes by is a state, each move is an action, each point received after a move is a reward. The RL consists in an AI playing this game until finding the best possible way to win, we can change the maze, the initial position and etc., but for explaining this is enough. We are going to use this toy example to explain Q-learning and later deep Q-learning. For more reference there is the work of (ZAFRANY, 2016) which explains and gives the code for an AI using deep-RL to solve more complicated mazes.

2.2.4 Q-Learning

Q-Learning is a model-free RL algorithm, and its goal is to learn the value of being in a given state and taking an action there. Usually, it has a table which has as rows the states and the actions as columns, in our case it would be a  $12 \times 4 = 48$  pairs of state-action. In fact, we could

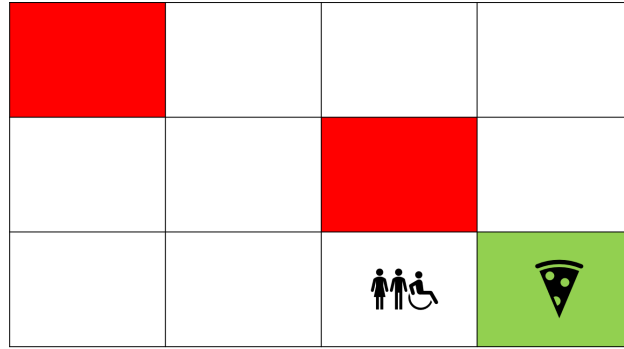


Figure 2.10: Maze: Rewards. Where  $j$  is the number of moves used by the player.

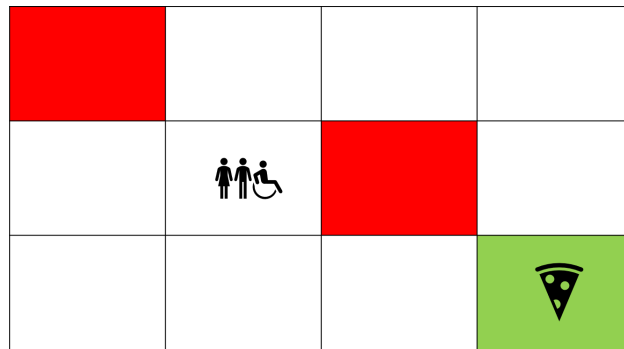


Figure 2.11: Maze: Rewards. Where  $j$  is the number of moves used by the player.

simplify since there are three terminal states (the ones that once the agent goes there it is game-over), and since when the agent is at the sides of the grid it has only 2 or 3 possible actions. The table is initialized with only zeros. Later, when receiving the rewards we will update the table accordingly using an equation based in the Bellman Equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r + \gamma \max_{a'} (Q(s_{t+1}, a'))] \quad (2.7)$$

where  $s_t$  is the current state and  $a_t$  the action that the agent chooses to take in  $s_t$ .  $Q(s_t, a_t)$  is the function which estimates the quality of a move,  $\alpha$  is the learning rate, which controls how fast we change our estimations,  $r$  is the reward you get from taking the action  $a$  in the state  $s$ .  $\gamma$  is the discount factor, which controls the importance of the future rewards, and  $\max_{a'} (Q(s_{t+1}, a'))$  which is the maximum expected future reward given the new state  $s_{t+1}$  and all the possible new actions  $a'$ .

Defining the positions and reward as defined in Figure 2.15, we have our initial Q-table as Table 2.1.

Taking  $\alpha = 1$ ,  $\gamma = 0.99$ . If we move to the right, Figure 2.9, the value  $[1, A] \times R$  will be

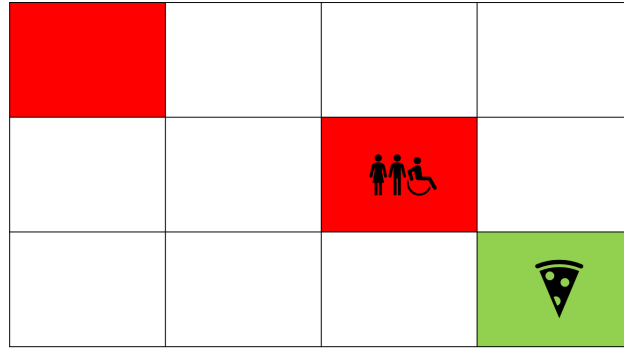


Figure 2.12: Maze: Rewards. Where  $j$  is the number of moves used by the player.

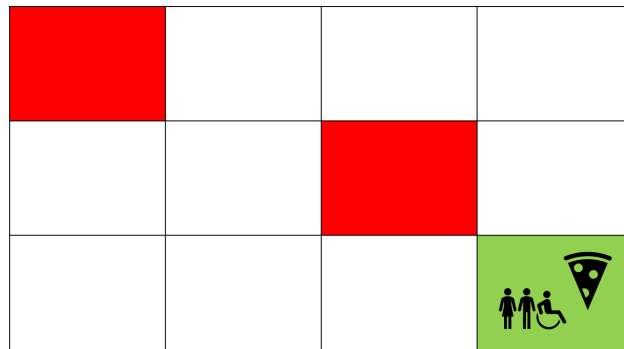


Figure 2.13: Maze: Rewards. Where  $j$  is the number of moves used by the player.

updated to  $Q(s_t, a_t) = 0 + 1 * [1 + 0.90 * 0] = 1$ . Similarly, if go to the right again the value  $[1, B] \times R$  will be updated to 1 and finally if we go to the right again, the value  $[1, C] \times R$  will be updated to 9. This will result in the Table 2.2

If we repeat this a second time the table will be updated as follows:

- $[1, A] \times R = 1 + 1 * [1 + 0.90 * 1] = 2.90$
- $[1, B] \times R = 1 + 1 * [1 + 0.90 * 9] = 10.10$
- $[1, C] \times R = 9 + 1 * [9] = 18$

This will result in the Q-table represented in 2.3.

So, we would use a random algorithm to explore and update our Q-table. At some point the Q-table will have the maximum values in the shortest path, or at least it should. One has to remember that Q-learning, as well as any RL algorithm, has two dilemmas.

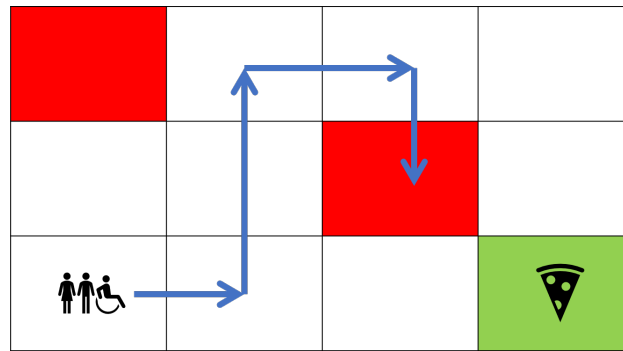


Figure 2.14: Maze: Rewards. Where  $j$  is the number of moves used by the player.

	A	B	C	D
3	-50	1	1	1
2	1	1	-50	1
1	-10	1	1	$15-2*j$

Figure 2.15: Maze: Rewards. Where  $j$  is the number of moves used by the player.

## 2.2.5 RL Dilemmas

### 2.2.5.1 Exploration x Exploitation

Our toy problem illustrates this dilemma well. Luckily, we took the shortest path first, so if the policy of our algorithm is to always take the action with the maximum Q-value, it will always take the shortest path if it uses the Q-table in the Table 2.2 or Table 2.3. But, what if it had taken the longest way, the exact opposite would occur and it would never learn the shortest path. So every RL-algorithm is a compromise between exploration and exploitation, i.e., in the beginning it will have a higher probability of choosing randomly its path (exploration phase) and as times goes on it will favor the paths with highest Q-value (exploitation phase). This will be defined by  $\epsilon$ , which is a function that decreases over time.

$$\epsilon = 1 - \delta * j$$

where  $j$  represents the number of moves of the agent and  $\delta$  the decaying factor of  $\epsilon$ . And a possible policy would be:

	U	D	L	R
[1,A]	0	N	N	0
[1,B]	0	N	0	0
[1,C]	0	N	0	0
[1,D]	N	N	N	N
[2,A]	0	0	N	0
[2,B]	0	0	0	0
[2,C]	N	N	N	N
[2,D]	0	0	0	N
[3,A]	N	N	N	N
[3,B]	N	0	0	0
[3,C]	N	0	0	0
[3,D]	N	0	0	N

Table 2.1: Q-table, where  $N$  represents the impossible moves.

	U	D	L	R
[1,A]	0	N	N	1
[1,B]	0	N	0	1
[1,C]	0	N	0	9
[1,D]	N	N	N	N
[2,A]	0	0	N	0
[2,B]	0	0	0	0
[2,C]	N	N	N	N
[2,D]	0	0	0	N
[3,A]	N	N	N	N
[3,B]	N	0	0	0
[3,C]	N	0	0	0
[3,D]	N	0	0	N

Table 2.2: Q-table, where  $N$  represents the impossible moves.

- Choose a random number  $N$  in the interval  $[0, 1[$
- If  $N > \epsilon$  the agent takes the action with the highest Q-value
- If  $N < \epsilon$  the agent chooses a random direction

A policy like this would ensure more exploration in the beginning and later it would favor the best paths but still explore the grid.

#### 2.2.5.2 Defining rewards

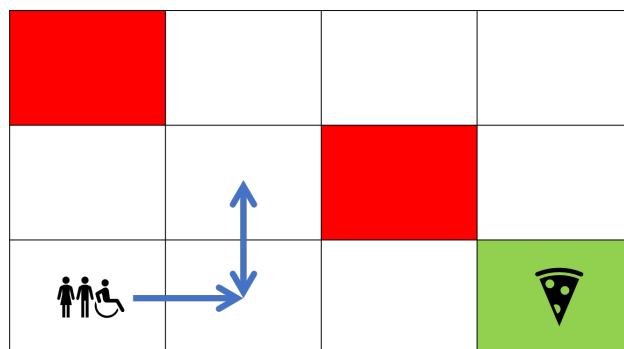
We purposely specified a bad reward. This with the objective of exemplifying one of the problems of RL solutions. The path to maximize the reward is to repeat until game-over by number of moves the squares with reward equals to 1, as example Figure 2.16. If we calculated the score in

	U	D	L	R
[1,A]	0	N	N	2.90
[1,B]	0	N	0	10.10
[1,C]	0	N	0	18.00
[1,D]	N	N	N	N
[2,A]	0	0	N	0
[2,B]	0	0	0	0
[2,C]	N	N	N	N
[2,D]	0	0	0	N
[3,A]	N	N	N	N
[3,B]	N	0	0	0
[3,C]	N	0	0	0
[3,D]	N	0	0	N

Table 2.3: Q-table, where  $N$  represents the impossible moves.

the end of the game, it would be  $1 * 15 = 15$  which is bigger than the 11 obtained using the shortest path. So, with our current reward system, the AI could learn to have an agent to wander until the time is out, and it would, hopefully, learn to do so.

This is the reason why choosing a reward is so complicated. It can lead to unintended and unwanted behaviors. For a maze problem, usually, you would give a negative reward for each move, this is to avoid the wandering around phenomenon. We also would not need reward in every square, only in the ones with terminal state and so on. Take a look in (ZAFRANY, 2016), there he shows a perfect example of how to solve a maze using RL. For our maze it would be enough to reduce the number of moves to 10, and it would be a good but not scalable or generalized reward.

Figure 2.16: Maze: Rewards. Where  $j$  is the number of moves used by the player.

### 2.2.6 Deep Q-Learning

Our toy example is really simple, therefore, we can build a Q-table containing all the possibilities. But, what if we could not build a Q-table because there are a really large number of



states or even an infinite number of states? To overcome this problem, researchers have been using Neural Networks (*NN*) to approximate the Q-table.

As defined by (SKYMIND, 2019), "Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns". With the improvements in computing power, they have been gaining space in the a wide range of ML tasks, such as image classification (SUN et al., 2019), speech recognition (HUANG et al., 2019) and (OORD et al., 2019), and even game playing/intelligence (KATONA et al., 2019) and (PATEL et al., 2019a). Its uses have been expanded to other areas, as robotics (GENG et al., 2019) and (PATEL et al., 2019b). Our previous definition gives an idea about the concept behind neural networks, but not of its function. In (GROHS et al., 2019) they tell us that "deep networks are optimal approximators for vastly different function classes", in fact neural networks are the approximation of functions and this is the definition that the reader should retain: **Neural networks can be seen as function approximators.**

It is this quality of the NN that we will be exploring. If we can not build a Q-table we can use a Neural Network to approximate our Q-table. So every time we need the Q-value we will use the NN to obtain it, and everything else will be the same as defined before for Q-learning, and this will result in our deep Q-learning.

## **CHAPTER III**

### **Methodology**

### 3.1 Case Study Implementation

We have seen both, the aeroelastic problem definition and the reinforcement learning definition. In this section we will see how both of them are put together to create an AI capable of solving our problem.

We have a *simulator* code, responsible for simulating the wing response, in other words, the code based in the aeroelasticity equations. The binary flutter system, represented by the surface heave and pitch motions, has the most common coupling modes of aircraft wings and here it can be seen as a clamped flexible wing of an aircraft.

---

#### Algorithm 1 Aeroelasticity code

---

**Require:**  $V, m, \mathbf{y}_0, s, c, x_f, x_{cm}, e, a_w, EC, \rho, \beta, t_g, amp_g$

**Ensure:**  $\mathbf{y}, \mathbf{g}, \mathbf{h}, \mathbf{w}_g, \beta$

1: Initializes all required parameters

2: Calculates the matrices

- Inertial Matrix  $\mathbf{A}$ ,
- Stiffness Matrix  $\mathbf{E}$
- Aerodynamic damping matrix  $\mathbf{B}$
- Aerodynamic stiffness matrix  $\mathbf{C}$
- Structural Damping  $\mathbf{D}$

3: Generates  $\mathbf{w}_g$

► 1-cosine

4: Calculates ODE matrices

- $\mathbf{Q}$
  - $\mathbf{g}$
  - $\mathbf{h}$
- 

where  $V$  is airplane velocity,  $m$  is airplane mass,  $\mathbf{y}_0$  are the airplane initial conditions,  $s$  is the semi-span,  $c$  is the chord,  $x_f$  is the elastic axis position,  $x_{cm}$  is the center of mass position,  $e$  is the eccentricity between elastic axis and aerodynamic center,  $a_w$  is the lift curve slope,  $EC$  is the fraction of chord made up by control surface,  $\rho$  is the air density, supposed constant,  $\mathbf{y}$  are the airplane positions and velocity (in coherence with the normalized coordinates),  $\beta$  is the angle between the control surface and the wing,  $t_g$  is the duration of the gust and  $amp_g$  the amplitude of it and  $\mathbf{w}_g$  is the vector containing the velocity of the upward wind in the z-direction for each time step. Since in our problem the airplane required parameters as mass, velocity and other are

constant, there is no need to recalculate this for the simulation time (one loop).

---

**Algorithm 2** EDO evolution and reward
 

---

**Require:** Algorithm 1 ( $y, g, h, w_g, \beta$ ), Controller ( $action$ )

**Ensure:**  $y, r, d$

► **Preparation**

1: Receives and stores all received parameters from Algorithm 1

► **Steps**

2: **while**  $d \neq 1$  **do**

3:    $i \leftarrow i + 1$

4:   Receives  $action$

► This is the action chosen by the controller

5:    $\beta \leftarrow \beta + action$

6:    $y \leftarrow$  4th order Range-Kutta results

► This simulates one step based on the action taken

7:   Calculates the reward  $r$

► **Ending conditions**

8:   **if**  $i \geq 1000$  or other ending condition **then**

► For example, the wingposition or velocity

crosses a limit

9:     $d \leftarrow 1$

10:   **end if**

11: **end while**

---

where  $r$  is the reward,  $d$  is a binary number stating the end of the simulation,  $i$  is the counter.

**Algorithm 3** Reinforcement Learning**Require:**  $N_{steps}$ ,  $N_{episodes}$ ,  $N_{sample}$ ,  $N_{transfer}$ ,  $\epsilon$ , Algorithm 2**Ensure:**

```

1: Initialize replay memory
2: Initialize action-value function  $Q$  with random weights  $\theta$ 
3: Initialize  $\hat{Q}$ ,  $\hat{\theta} \leftarrow \theta$ 
4: for  $steps = 1, N_{steps}$  do ► Steps
5:   for  $Episode = 1, N_{episodes}$  do ► in the end of the simulation, i.e.,  $d = 1$  the episode is
     reset ► Episodes
6:     if  $n_{random} < \epsilon$  then ► Exploration
7:        $action \leftarrow$  random action
8:     else ► Exploitation
9:        $action \leftarrow \operatorname{argmax}_a Q(s)$ 
10:    end if
11:    Plays one step of Algorithm 2 and gets  $r, \mathbf{y}, d$ .
12:     $s_{t+1} \leftarrow s_t, action_t, \mathbf{y}$ 
13:    Stores  $s_{t+1}$  in replay memory
14:    if  $steps = N_{sample}$  then ► Training
15:      Sample random minibatch from replay memory
16:      Train  $Q$  and actualizes weights  $\theta$ 
17:    end if
18:    if  $steps = N_{transfer}$  then ► Weight transfer
19:      Actualizes  $\hat{Q}, \hat{\theta} \leftarrow \theta$ 
20:    end if
21:  end for
22: end for

```

## 3.2 Numerical Simulations

All code used in this work, this text itself and everything necessary to reproduce the current results are can be found in the authors github<sup>©</sup>, link given in the footnote<sup>1</sup>.

### 3.2.1 Rewards

In order to find an good reward, we passed by six different rewards systems, which can be found in our source code<sup>2</sup>. Here we presented only three of them. Because we considered the most relevant ones.

#### 3.2.1.1 R1

The first reward was a simple idea, we wanted the value of  $q$  not to increase overtime, so:

$$R1 = \text{Sum}(|q_t| - |q_{t-1}|) / 2 \quad (3.1)$$

where *Sum* represents a sum of all vector elements. It is important to know that the AI tries to minimize the reward. The intention in this reward was to minimize the variation of the oscillation. The result with the network presented in 5.1 is seen in the Figures 3.1 and 3.2.

<sup>1</sup>Link to the codes: <https://github.com/rassbr/AI-techniques-applied-to-alleviate-wind-gust-in-airplanes>

<sup>2</sup>Link to rewards' code: <https://github.com/rassbr/AI-techniques-applied-to-alleviate-wind-gust-in-airplanes/blob/master/reward.py>

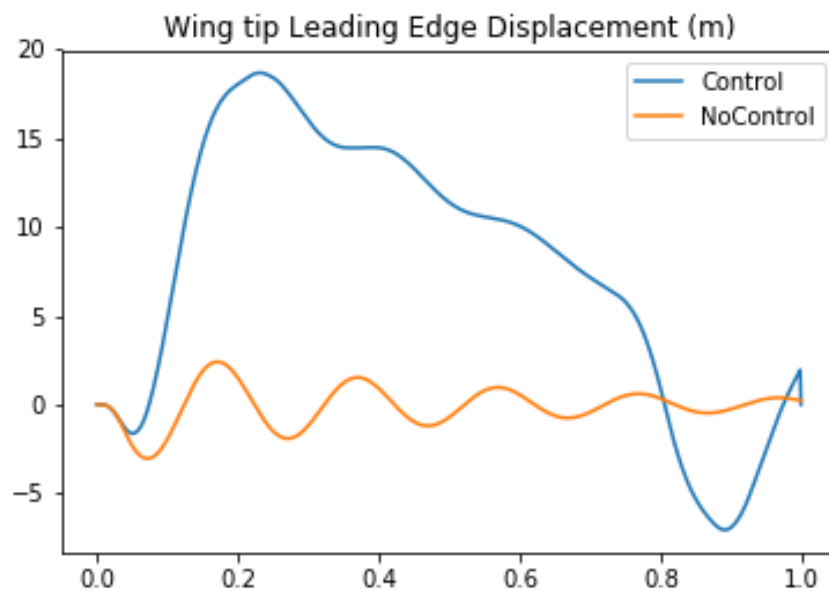


Figure 3.1: Wing tip lead displacement in meters of a network trained with the reward R1, given a result score of 408.

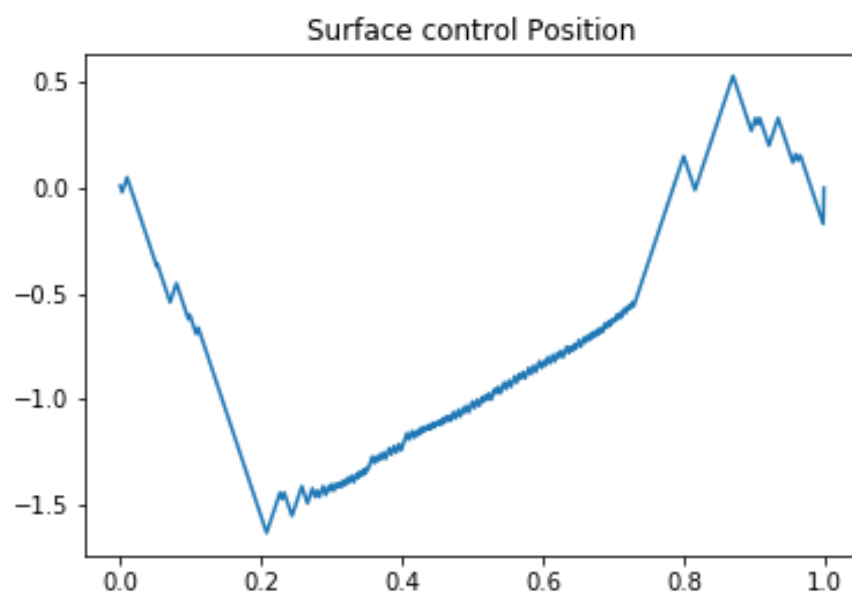


Figure 3.2: Surface control position of a network trained with the reward R1, giving a result score of 408.

We have an ill-behavior, which gives a result that we can clearly see that it does not minimize the effects of the aeroelastic phenomenon, in fact, it is better to have no control in this case because the system alone is capable of reducing the gust effect. Nonetheless, the score (sum of all rewards) is higher for this solution than to the no control solution, which means that this reward does not

express the intended behavior and have to be discarded. In a short term it does not guide the AI agent in the *right* direction. This means, that no further analysis or test will be done with it.

This is a good example of the difficulties of using RL for real-problems, the reward is not trivial, and it can result in ill-behaviors, making the process of shaping the reward a critical phase of the project.

### 3.2.1.2 R4

This reward,  $R4$ , was based in using a referential. It was thought that if we used the no control solution as reference to give the reward, it could improve itself near to this solution initially. The equation would be like this:

$$R4 = |z_{ref}| - |z|. \quad (3.2)$$

where  $z_{ref}$  is the  $z$  of the no control condition. We also put a penalty if it trespassed the game-over condition. From this reward we didn't obtain good results. Training it for 70 games gave the following result:

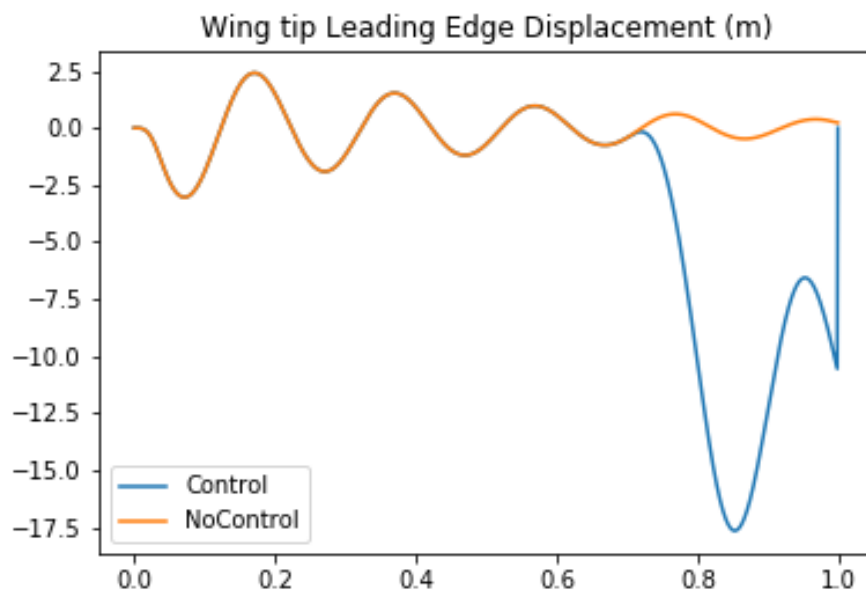


Figure 3.3: Wing tip leading edge displacement in meters of an AI agent trained with the reward  $R4$ .



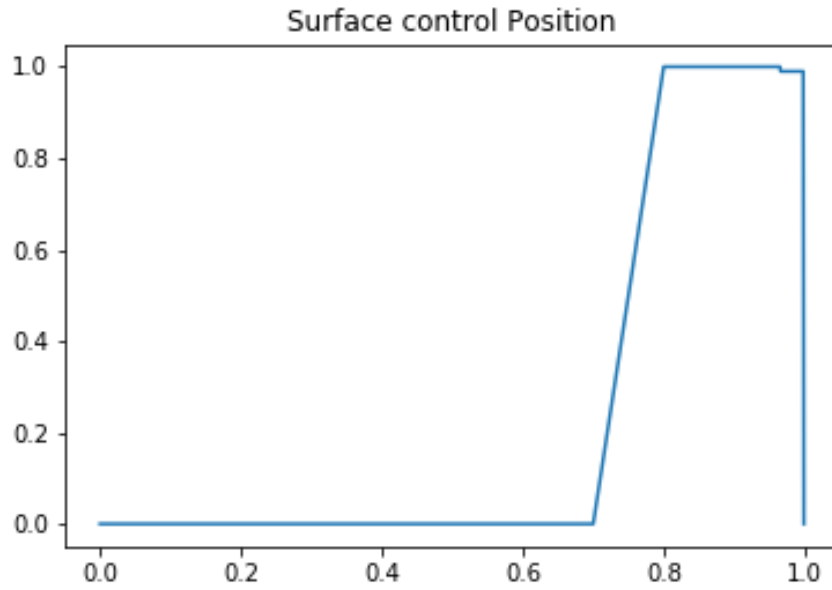


Figure 3.4: Surface control position of an AI agent trained with the reward R4.

which is clearly not a good solution. Training more made the solution equals to the no control solution, therefore we discarded it.

### 3.2.1.3 R6

This reward intends to reduce the wing tip displacement  $z$ :

$$R6 = \frac{1}{|z|}. \quad (3.3)$$

The problem of this reward is when  $z$  is close to zero the value tends to the infinity, which is not desirable. So, in order to mitigate this effect, we limited its value to 5. Besides, to motivate the AI to keep playing we limited the smallest reward to 0.1. So we have a range of rewards between  $.1 \leq R6 \leq 5$ , these limits have, obviously, negative effects. Any value of  $z$  smaller than 0.2 receives the same reward and and value bigger than 10 do also receive the same reward, so the AI agent could stop improving because it does not see any improvement.

In Figure 3.5 it is showed the result after 65 games for the wing tip displacement,

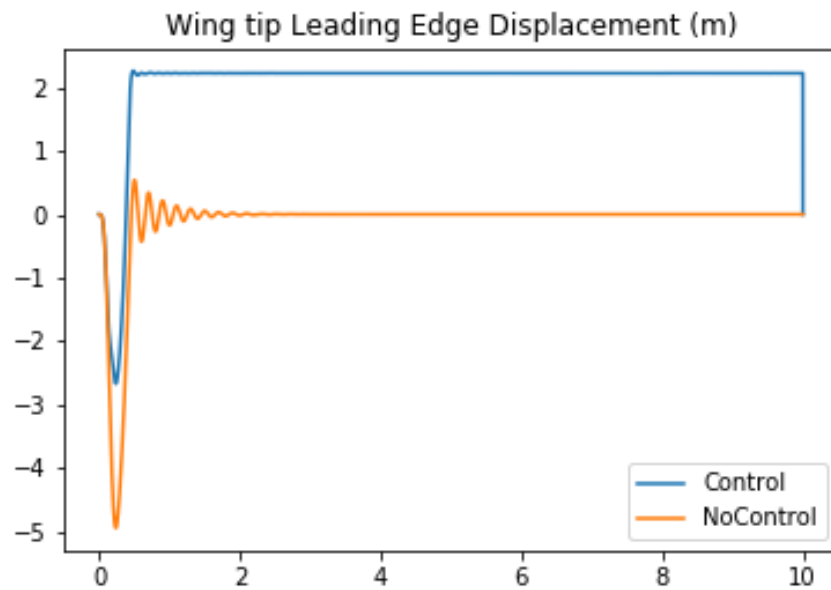


Figure 3.5: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 5038.76.

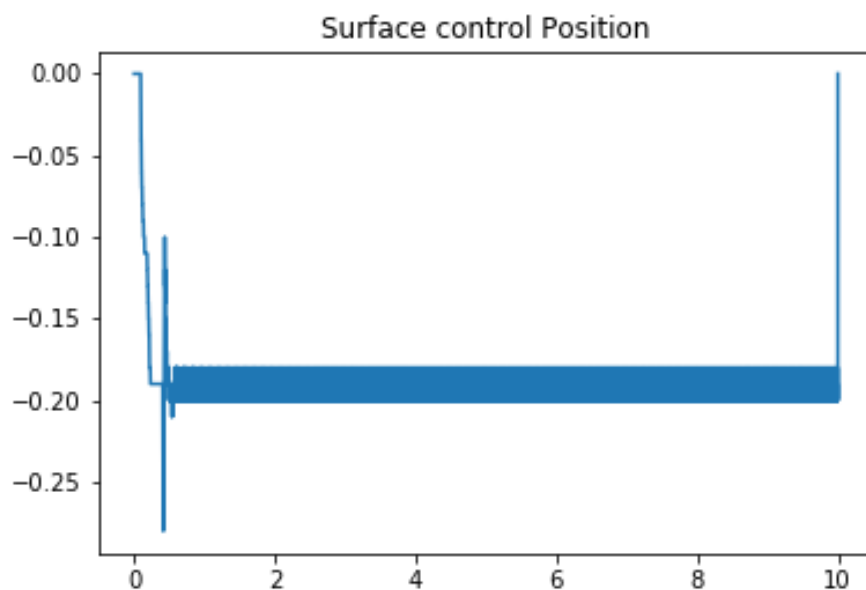


Figure 3.6: Surface control position of an AI agent trained with the reward R6, giving a result score of 5038.76.

This result has two advantages, the first its absolute biggest value is smaller than the one with no control and it seems to stabilize near the equilibrium point (close to the value of 2) faster than its counterpart. Nonetheless, if we look to the Figure 3.6, it is possible to see that the control surface does not stop moving, so it has not learnt to stop moving the control surface position. This

is not essentially bad, once we didn't penalise moving the control surface.

### 3.2.2 Neural Networks

Finding the good NN was not trivial, we started from an NN as small as 5 layers only and kept increasing it and changing the number of units (neurons) in each layer. This was to see if we should have a deeper network but with fewer units in each layer or larger but fewer layers. As it will be shown next, it is closer to a deep network with fewer units in each layer. We also tried using some *tricks* to help our NN to learn, for example, dropouts, which are connections which are randomly cut during the training to make a robuster NN, but in the end the result didn't seem qualitatively better, so we stayed with the simplest one.

Another important point is that we used a fully connected neural network (only dense layers), which means that all units from a previous layer are connected with all units with the next layer. This choice seemed the most reasonable one because it is simpler than other neural networks as recurrent neural network or *convolutional neural networks* (CNN). Nonetheless, we did not use CNN for another reason, since we are not working with images there is no real fundament to implement it.

The final neural network, hereon called *NN1*, used in all previous case for analysis is the one presented in Table 5.1. Which is a completely dense NN composed by seventeen layers, which starts with 9 units, increases and keeps it values in 18 units before finally reducing to 9 units and then only 3, which represents the number of different actions that can be chosen by our network. The question that may remains is, "is this the best neural network we can have?", the answer is probably not. After obtaining the previous results, we tried to simplify it, and the result is the network present in Table 5.2, hereon referred as *NN2*, which only seven layers. The results between both will be compared in the next section.

For simplicity's sake, we will only present another NN that is the one used for flutter, Table 5.3, but it did not give good results for this case, as shown in future works, neither was necessary for the velocities bellow flutter.

## **CHAPTER IV**

### **Results**

## 4.1 Results analysis and evaluation

### 4.1.1 Analysis

It could be seen that R6 is a satisfying reward with the neural network presented in Table 5.1. So we will further develop its analysis. In Figures 3.5 and 3.6 we saw the result for the AI agent trained in 65 games. But what if we let it train further, for example, 118 games? The result would be the one presented below:

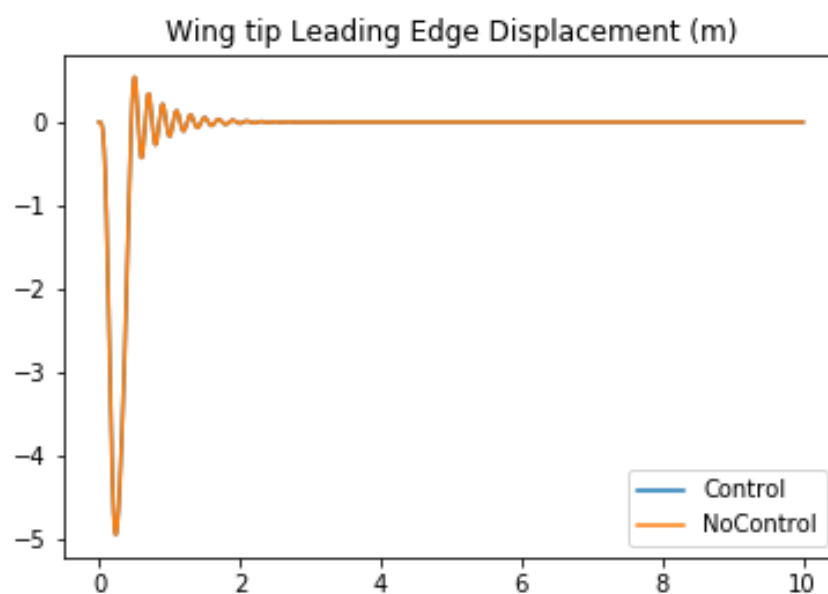


Figure 4.1: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 47882.96.

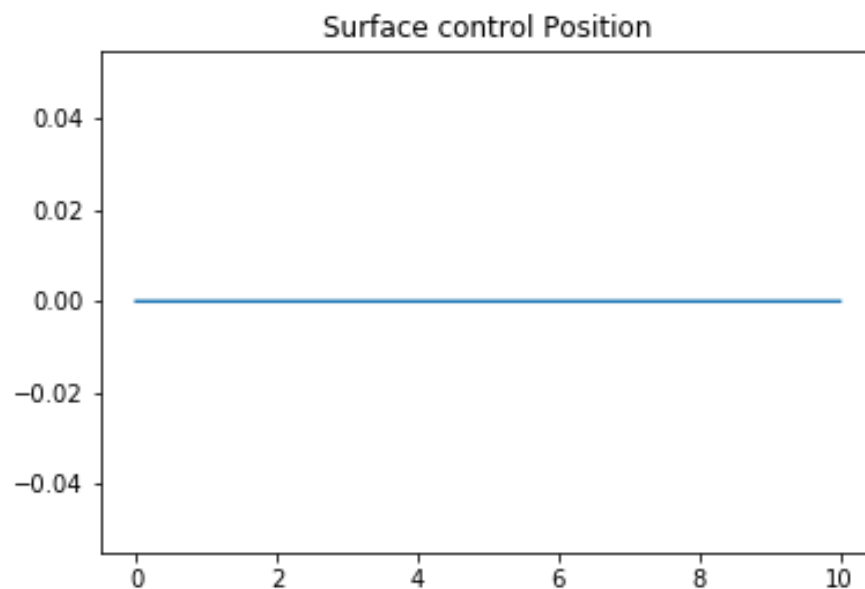


Figure 4.2: Surface control position of an AI agent trained with the reward R6, giving a result score of 47882.96.

We could say that in one aspect this result is expected, since its score is bigger than the previous one, i.e., the AI learned how to maximize its score. Nonetheless, this solution is the result of doing nothing, i.e., equal to the no control solution. If we were to try to other velocities, what we have done, it will always do nothing.

Maybe 118 games were too much, what if we trained only two more games, i.e., a total of 67 games played? The result would be the following:

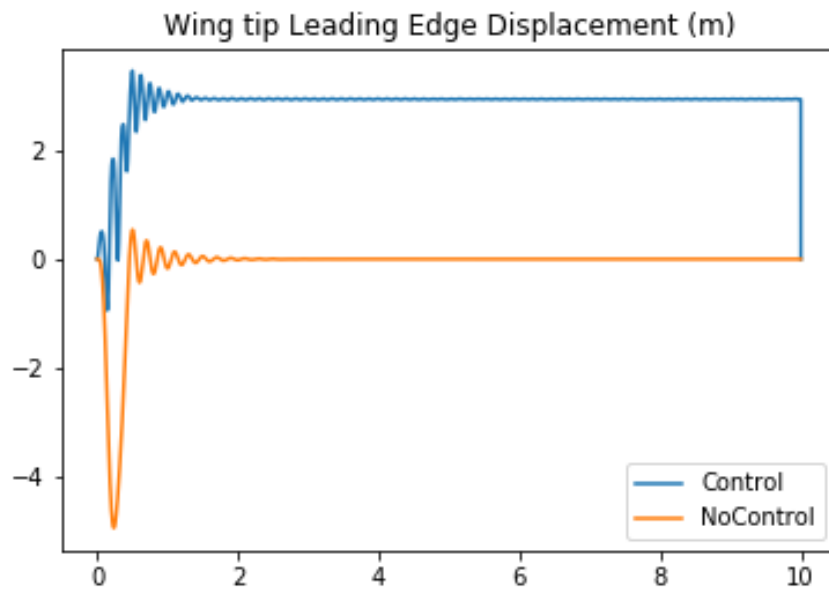


Figure 4.3: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 4126.34.

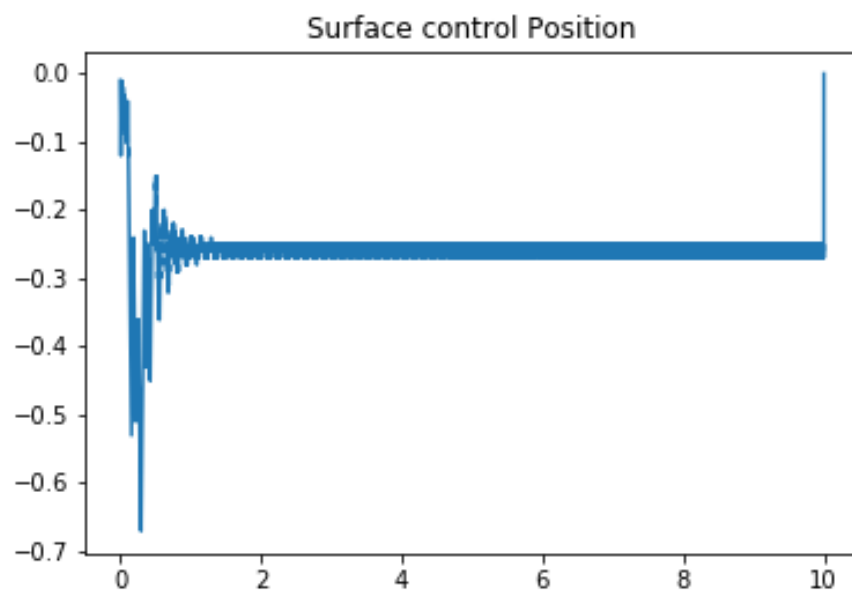


Figure 4.4: Surface control position of an AI agent trained with the reward R6, giving a result score of 4126.34.

Like the one with 65 games, it does not stop moving the control surface. It completely mitigates the first valley, nonetheless it takes more time to stabilize in the equilibrium position. Is this solution better than the one presented in Figure 3.5 (65 games)? We can not say for sure, because it depends on the criteria used. This will be discussed in the following section *Evaluation*.

Until now we have been seeing the result given a velocity of 100m/s, however an airplane flies with different velocities, making it necessary to be able to alleviate the gust for different velocities. So, we will show the results of the agent trained in 65 games for other velocities randomly chosen.

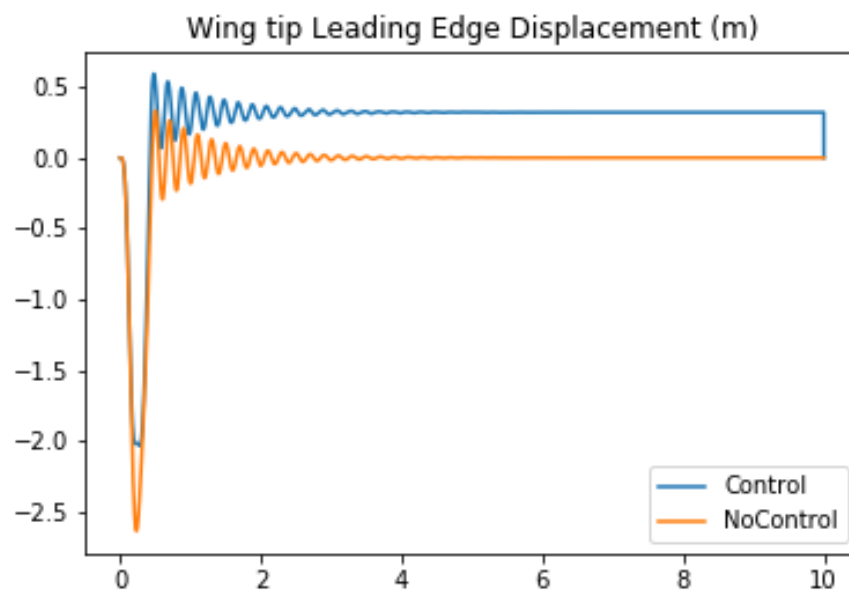


Figure 4.5: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 30901.44, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s.



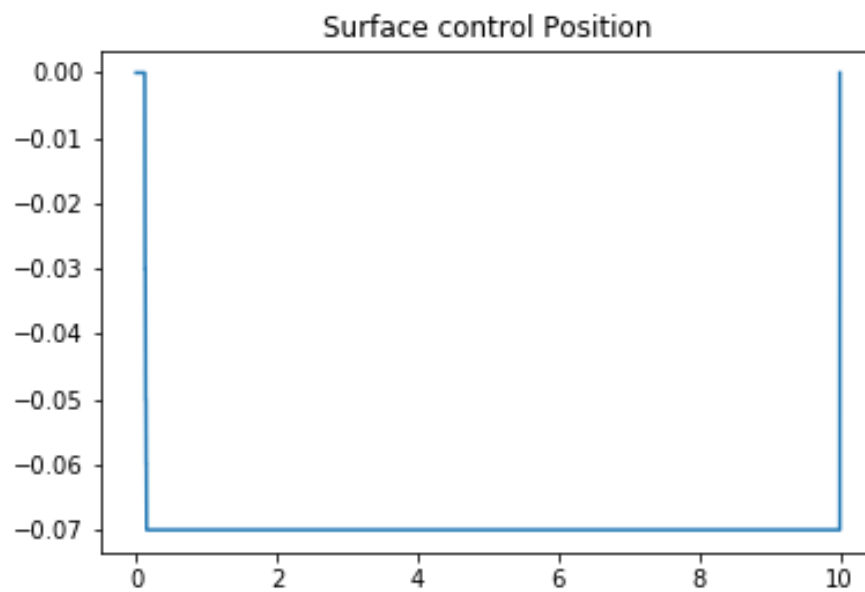


Figure 4.6: Surface control position of an AI agent trained with the reward R6, giving a result score of 30901.44, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s.

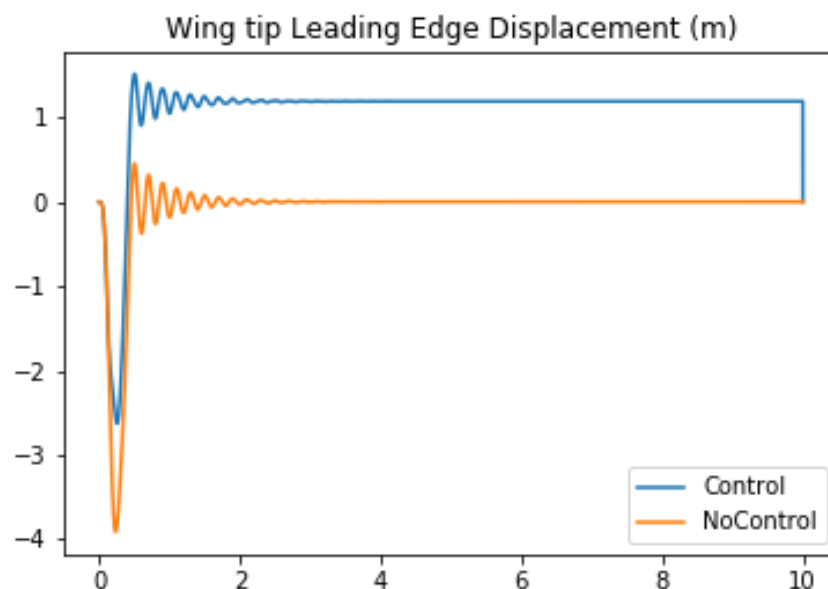


Figure 4.7: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s.

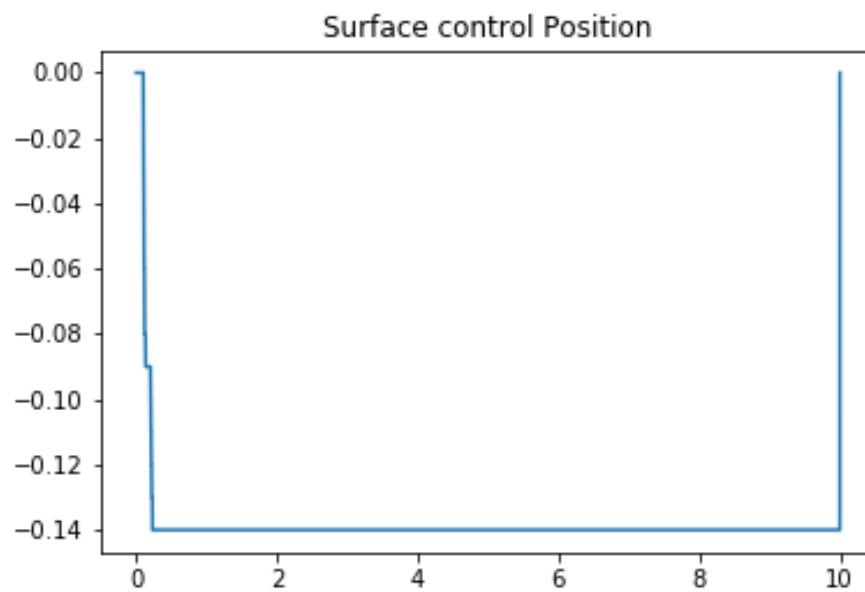


Figure 4.8: Surface control position of an AI agent trained with the reward R6, giving a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s.

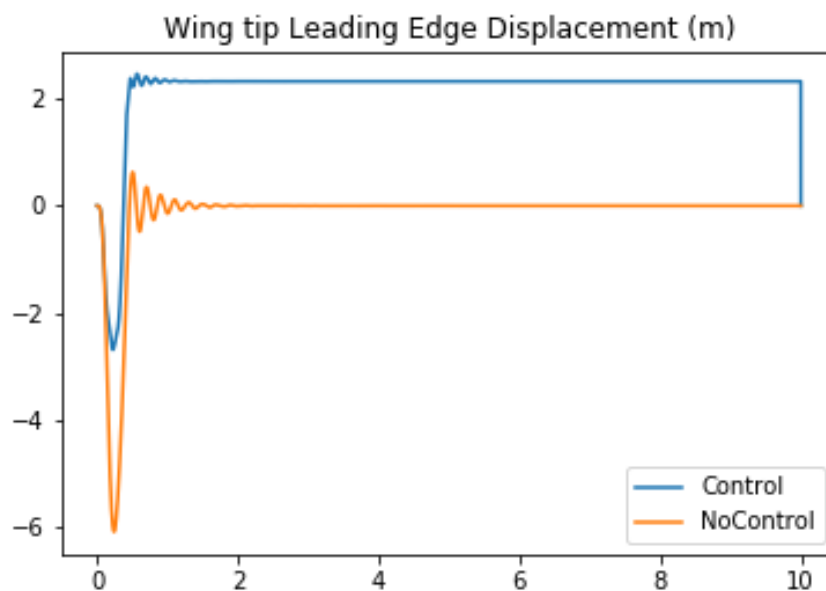


Figure 4.9: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of 4803.39, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s.

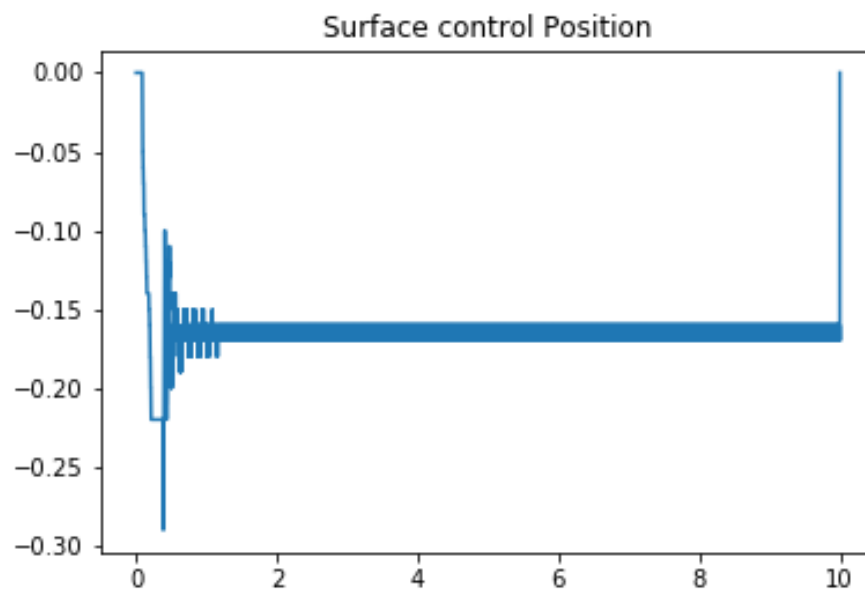


Figure 4.10: Surface control position of an AI agent trained with the reward R6, giving a result score of 4803.39, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s.

In the three cases, Figures 4.5 to 4.10, the AI agent gives a satisfactory solution, because it reduces the amplitude of the first valley maintaining or reducing the time until it stops oscillating.

#### 4.1.1.1 NN1 vs NN2

We have between Figures 4.11 to 4.18 the equivalent figures that were done for NN1 made for NN2.

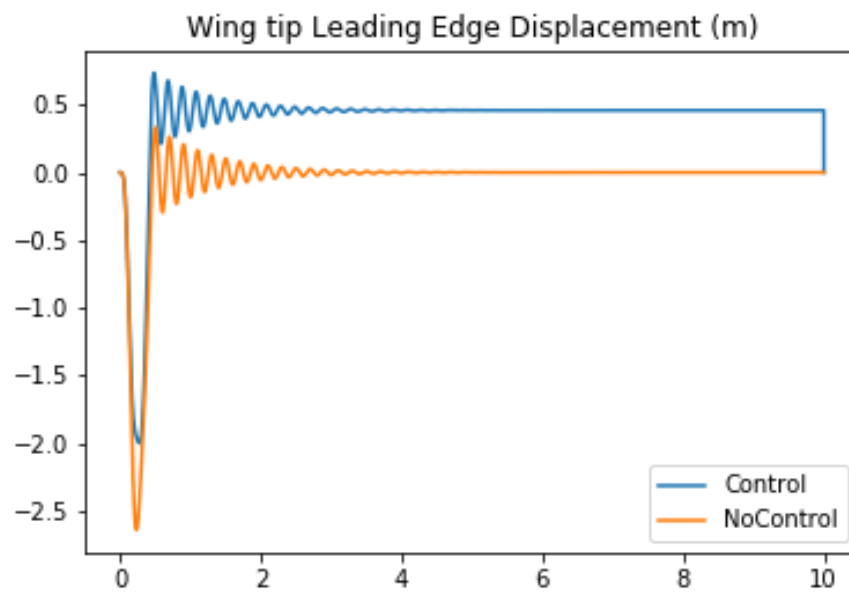


Figure 4.11: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and NN2, given a result score of 21895.78, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s.

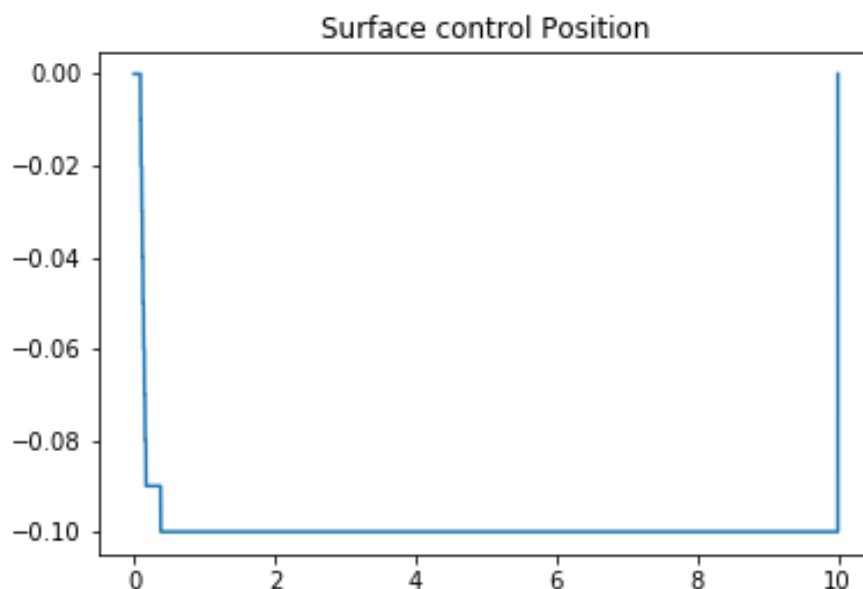


Figure 4.12: Surface control position of an AI agent trained with the reward R6 and NN2, giving a result score of 21895.78, while a no control solution gives a score of 48331.77. Considering the airplane velocity 59.03m/s.

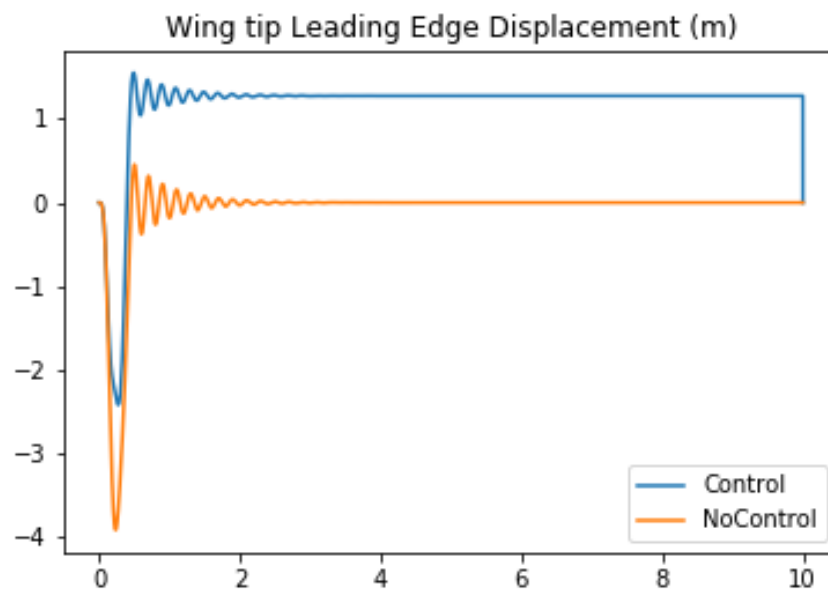


Figure 4.13: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and *NN2*, given a result score of 8271.52, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s.

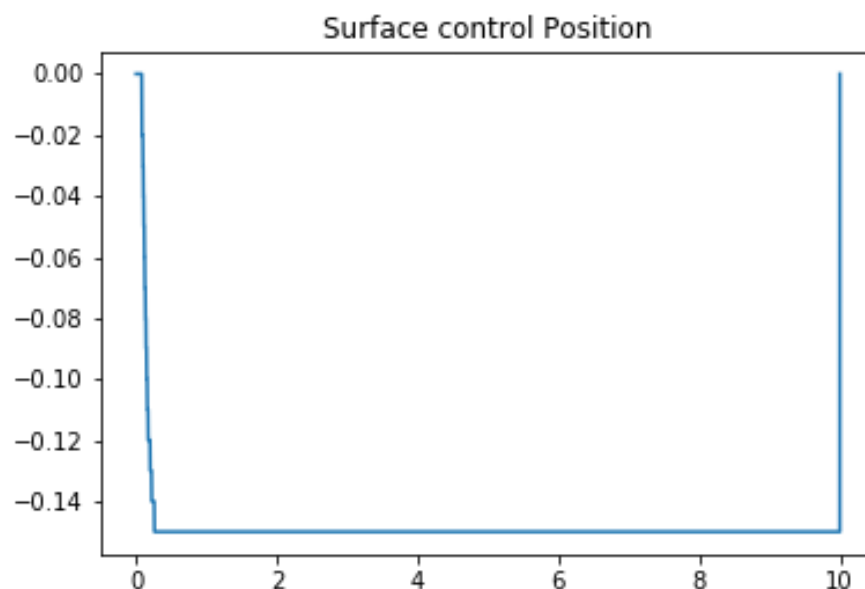


Figure 4.14: Surface control position of an AI agent trained with the reward R6 and *NN2*, giving a result score of 8271.52, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s.

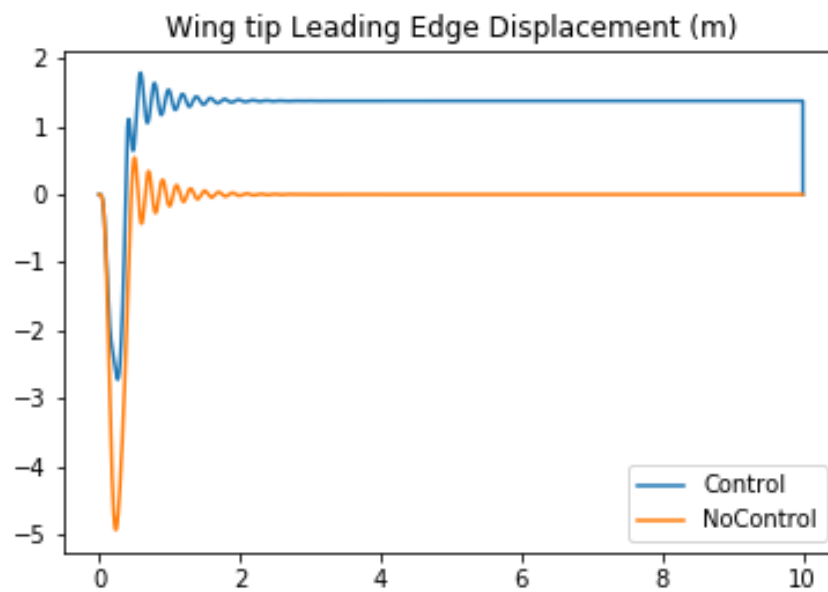


Figure 4.15: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and NN2, given a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s.

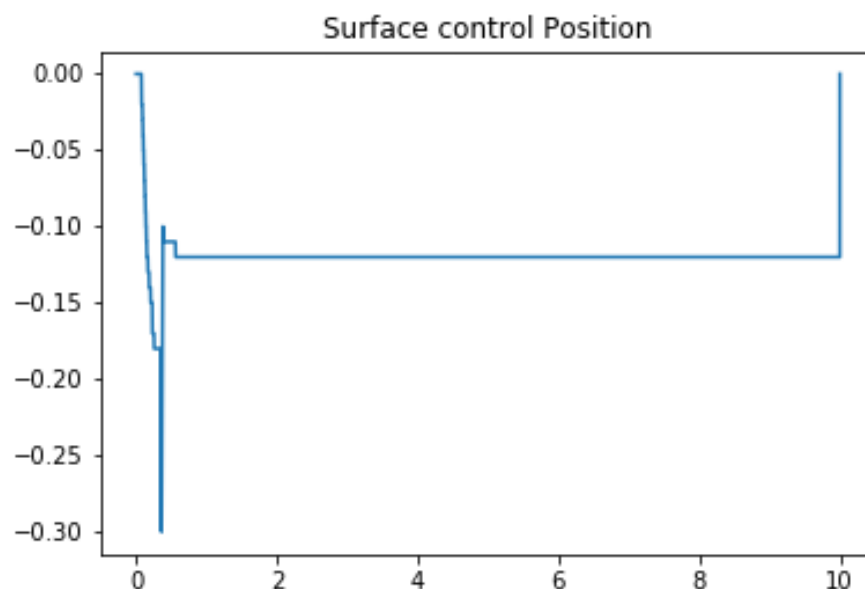


Figure 4.16: Surface control position of an AI agent trained with the reward R6 and NN2, giving a result score of 8819.93, while a no control solution gives a score of 48023.86. Considering the airplane velocity 83.10m/s.

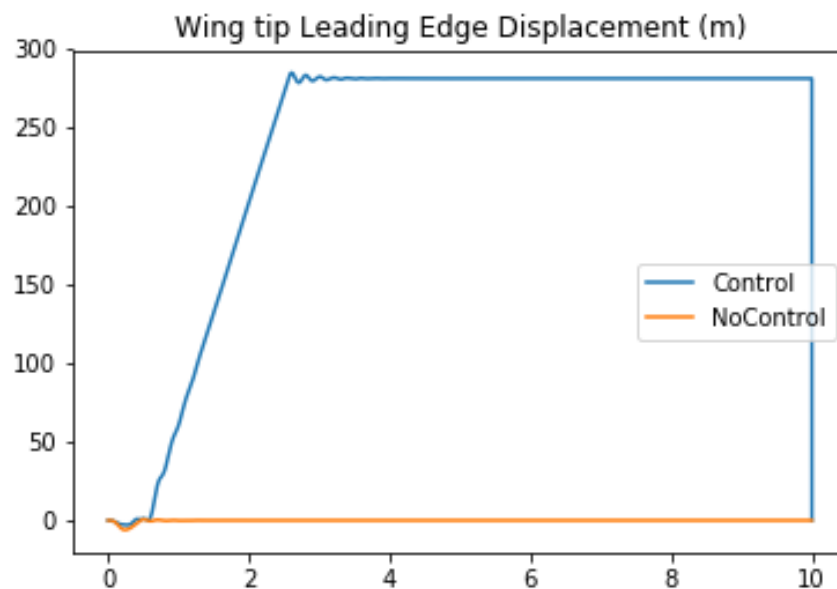


Figure 4.17: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6, given a result score of -92600.48, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s.

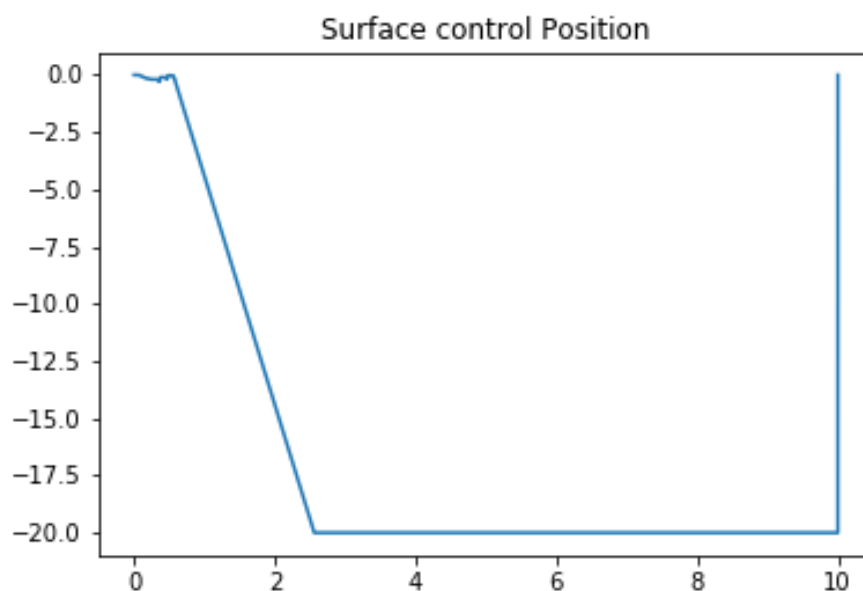


Figure 4.18: Surface control position of an AI agent trained with the reward R6, giving a result score of -92600.48, while a no control solution gives a score of 47787.03. Considering the airplane velocity 116.11m/s.

It is interesting to verify that for velocities 59m/s and 83m/s the behaviour for *NN1* and *NN2* was really similar. However for 100m/s it learned to stop moving the control surface after stabilizing, which can be considered a good result. Nonetheless, when we check the result obtained

for 116m/s we have that it diverges, meaning that this simplified neural network could not learn how to control faster velocities, i.e., it can only work in a smaller range of velocities, what could be explained by it having fewer parameters. We could try to suppose, also, that the behaviour of not stop moving the control surface is what helps to stabilise in faster velocities.

#### 4.1.2 Evaluation

We made a more qualitative analysis in the previous section, looking into the overall behavior, now we are interested in some measures to evaluate the results. We will have 3 criteria:

- C1: Overshoot of controlled system divided by the overshoot of uncontrolled system, being overshoot the maximum peak or valley in absolute values. The smallest the better.
- C2: Time to stabilize, considering stable any value between the final value with a margin of 3% of the Overshoot. The smallest the better.
- C3: Relative time to stabilize, time to stabilize of the controlled solution divided by the time to stabilize of the uncontrolled solution. The smallest the better.
- C4: Maximum delta of the controlled system divided by the maximum delta of the uncontrolled system, being maximum delta the variation between the highest valley and the lowest valley. The smallest the better.

Table 4.1: Performance evaluation of the results.

Performance Evaluation				
	C1 [%]	C2 [ms]	C3 [%]	C4 [%]
<b>G65 - V59</b>	77.19	1693	99.35	88.54
<b>G65 - V83</b>	67.22	1294	106.76	94.96
<b>G65 - V100</b>	54.01	455	44.78	89.96
<b>G67 - V100</b>	70.07	1032	101.57	80.31
<b>G118 - V100</b>	100	1016	100	100
<b>G65 - V116</b>	44.14	702	76.88	76.70

In the Table 4.1 we have  $GX$ , where  $X$  is a number representing the number of training games.  $VX$ , where  $X$  is a number representing the airplane velocity. From this table we can see that in every situation that the agent decides to act (give a command different from 0) we have



an smaller overshoot ( $C1 < 100\%$ ) and an smaller maximum delta ( $C4 < 100\%$ ). Nonetheless, in some cases the system takes more time to stabilize.

An interesting analysis here is that looking the caption of Figures 3.5 to 4.10, in all the cases when the agent acts the reward is smaller than the uncontrolled system. Nonetheless, given the criteria we defined, the results of our AI system playing are better than the uncontrolled system. It may seem contradictory, but in reality it only mean that our reward does not represents our criteria as it should. From this we may conclude that shaping the right reward is hard even when you have your criteria, but even an imperfect reward may give good results.

#### 4.1.2.1 NN1 vs NN2

Looking into the Table 4.2 we can verify that not *NN1* nor *NN2* gives the best result in all velocities. One interesting observation can be made, for criteria *C1*, *C2* and *C3* at 100m/s not having the oscillations in the control surface does not give better results, showing that, indeed, oscillations can be a good thing for performance in our case.

If we had to keep only one of the Neural Networks it would without a doubt be *NN1*.

Table 4.2: Performance evaluation of the results *NN1* vs *NN2*, highlighted in green the best result given for one velocity.

Performance Evaluation				
	C1 [%]	C2 [ms]	C3 [%]	C4 [%]
<b>G65 - V59</b>	77.19	1693	99.35	88.54
<b>Simplified - V59</b>	75.72	1694	99.41	92.02
<b>G65 - V83</b>	67.22	1294	106.76	94.96
<b>Simplified - V83</b>	61.77	1200	99.00	90.90
<b>G65 - V100</b>	54.01	455	44.78	89.96
<b>Simplified - V100</b>	55.15	1290	126.96	82.40
<b>G65 - V116</b>	44.14	702	76.88	76.70
<b>Simplified - V116</b>	-	-	-	-

# CHAPTER V

## Conclusions

### 5.1 Conclusion

In this work we presented a Reinforcement Learning technique, which is an already known one, applied in a new context, which is alleviating wind gust in airplanes. Firstly, it was explained and detailed the aeroelastic problem and the reinforcement learning, its problems and its potential. Secondly, the code that would be used itself was unraveled using pseudo-codes and a link<sup>1</sup> to access the codes was given. Thirdly, some key points, the network structure and the rewards used in our study case were shown. Finally, the results were analysed qualitatively and quantitatively (using 3 criteria).

We can conclude that the reinforcement learning may be applied to alleviate the wind gust in airplanes, in this work its result was validated for a small range of velocities, which does not include flutter velocities as it would be desirable. The contributions of this work are:

- Relating aeroelasticity and machine learning, showing that RL approaches, and possibly, other techniques which are in the trend nowadays may be used to solve aeroelasticity problems, such as gust alleviation.
- Making the code open-source and the methodology detailed and explained (considering the work here written and the available and commented coding in Jupyter Notebook, making it more humanized).

---

<sup>1</sup>Link to the codes: <https://github.com/rassbr/AI-techniques-applied-to-alleviate-wind-gust-in-airplanes>

- Validation of this approach for a range of velocities and showing the future possible work that can be build up over this one.

## 5.2 Future work

The first possibility for future works is extending this technique to the flutter condition. Even if its unconventional, here we will present what was done and some room for improvement. In this work we tried using the same reward and network and even a deeper network, but the results weren't good as it can be seen in Figures 5.1 and 5.2.

An even deeper neural network than the one presented in 5.3 (Appendix A), a more complex one, a recurrent NN or even working with the zone just before and just after flutter are four possibilities that could help to alleviate the wind gust in flutter conditions.

The second possibility would be training the network in different velocities so it could have a more homogeneous and optimised value for all the range. We tried this approach and it seems that it does not help the network to learn since every new game is too different from the previous one. An idea is using a pre-trained network for one velocity and trying to learn the *new* velocities after that.

The last possibility of future work here presented would be validating the method for different configurations of airplanes, of single NN for different aircraft.

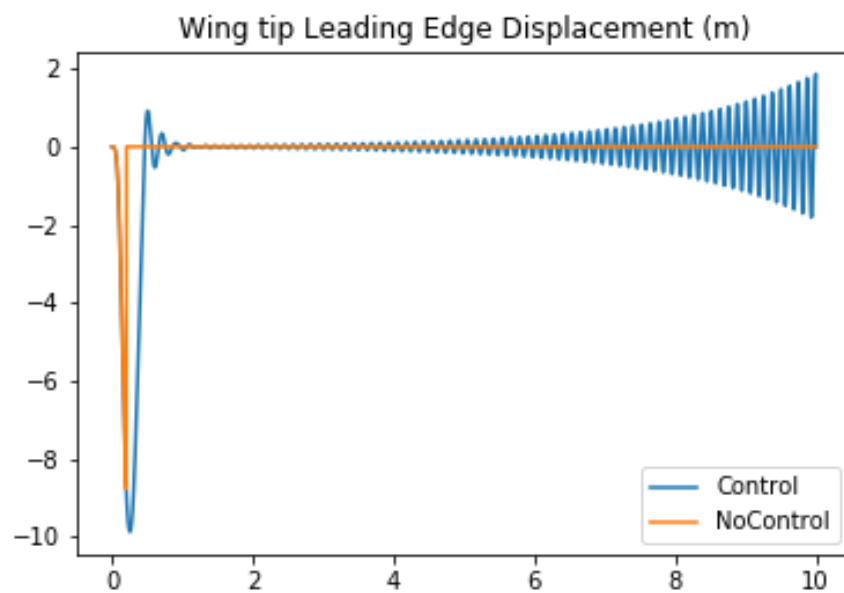


Figure 5.1: Wing tip leading edge displacement in meters of an AI agent trained with the reward R6 and a total of 100 games. Considering the airplane velocity 155m/s.

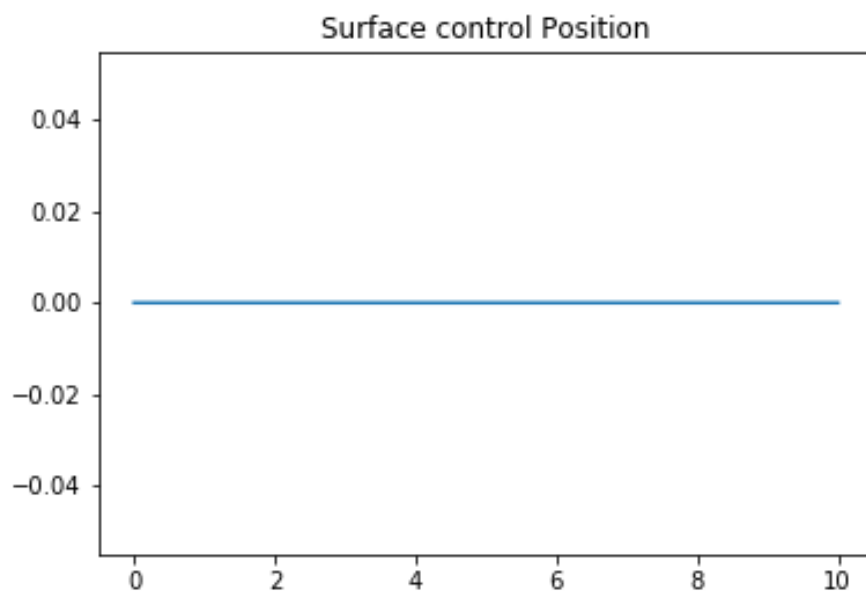


Figure 5.2: Control surface position of an AI agent trained with the reward R6 and a total of 100 games. Considering the airplane velocity 155m/s.

## Bibliography

- DOWELL, E. H. et al. *A modern course in aeroelasticity*. 5. ed. [S.l.]: Springer, 2015.
- GENG, M. et al. Learning to cooperate via an attention-based communication neural network in decentralized multi-robot exploration. *Entropy*, Multidisciplinary Digital Publishing Institute, v. 21, n. 3, p. 294, 2019.
- GOLLAPUDI, S. *Practical machine learning*. [S.l.]: Packt Publishing Ltd, 2016.
- GROHS, P. et al. *Deep Neural Network Approximation Theory*. 2019.
- HUANG, K.-Y. et al. Speech emotion recognition using deep neural network considering verbal and nonverbal speech sounds. In: IEEE. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2019. p. 5866–5870.
- KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996.
- KATONA, A. et al. *Time to Die: Death Prediction in Dota 2 using Deep Learning*. 2019.
- OORD, A. G. A. van den et al. *Speech recognition using convolutional neural networks*. [S.l.]: Google Patents, 2019. US Patent App. 16/209,661.
- PATEL, D. et al. Improved robustness of reinforcement learning policies upon conversion to spiking neuronal network platforms applied to atari games. *arXiv preprint arXiv:1903.11012*, 2019.
- PATEL, S. et al. An intelligent hybrid artificial neural network-based approach for control of aerial robots. *Journal of Intelligent & Robotic Systems*, Springer, p. 1–12, 2019.
- SKYMIND. *A Beginner's Guide to Deep Reinforcement Learning*. 2019. Date of Access: 10/06/2019. Disponível em: <<https://skymind.ai/wiki/deep-reinforcement-learning>>.
- SUN, Y. et al. Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, IEEE, 2019.
- SUTTON, R. S.; BARTO, A. G. *Reinforcement learning: An introduction*. [S.l.]: MIT press, 2018.
- WRIGHT, J. R.; COOPER, J. E. *Introduction to aircraft aeroelasticity and loads*. 2. ed. [S.l.]: John Wiley & Sons, 2008.
- ZAFRANY, S. *Deep Reinforcement Learning for Maze Solving*. 2016. Date of Access: 20/07/2019. Disponível em: <<https://www.samyzaf.com/ML/rl/qmaze.html>>.

Table 5.2: Neural Network simplified used for the AI agent, also with 65 games of training.

<b>Layer</b>	<b>(type)</b>	<b>Output Shape</b>	<b>Param (#)</b>
dense_1	(Dense)	(None, 9)	54
dense_2	(Dense)	(None, 18)	180
dense_3	(Dense)	(None, 18)	342
dense_4	(Dense)	(None, 18)	342
dense_5	(Dense)	(None, 18)	342
dense_6	(Dense)	(None, 9)	171
dense_7	(Dense)	(None, 3)	30
=====			
<b>Total</b>	<b>params:</b>	<b>1,461</b>	
<b>Trainable</b>	<b>params:</b>	<b>1,461</b>	
<b>Non-trainable</b>	<b>params:</b>	<b>0</b>	

### 5.3 Appendix A - Neural Networks used

Table 5.1: Neural Network used for the successful AI agent with 65 games.

<b>Layer</b>	<b>(type)</b>	<b>Output Shape</b>	<b>Param (#)</b>
dense_8	(Dense)	(None, 9)	54
dense_9	(Dense)	(None, 18)	180
dense_10	(Dense)	(None, 18)	342
dense_11	(Dense)	(None, 18)	342
dense_12	(Dense)	(None, 18)	342
dense_13	(Dense)	(None, 18)	342
dense_14	(Dense)	(None, 18)	342
dense_15	(Dense)	(None, 18)	342
dense_16	(Dense)	(None, 18)	342
dense_17	(Dense)	(None, 18)	342
dense_18	(Dense)	(None, 18)	342
dense_19	(Dense)	(None, 18)	342
dense_20	(Dense)	(None, 18)	342
dense_21	(Dense)	(None, 18)	342
dense_22	(Dense)	(None, 18)	342
dense_23	(Dense)	(None, 9)	171
dense_24	(Dense)	(None, 3)	30
=====			
<b>Total</b>			
<b>params: 4,881</b>			
<b>Trainable</b>			
<b>params: 4,881</b>			
<b>Non-trainable</b>			
<b>params: 0</b>			

Table 5.3: Neural Network used for flutter condition.

Layer	(type)	Output Shape	Param (#)
dense_1	(Dense)	(None, 9)	54
dense_2	(Dense)	(None, 18)	180
dense_3	(Dense)	(None, 18)	342
dense_4	(Dense)	(None, 18)	342
dense_5	(Dense)	(None, 18)	342
dense_6	(Dense)	(None, 18)	342
dense_7	(Dense)	(None, 36)	684
dense_8	(Dense)	(None, 36)	1332
dense_9	(Dense)	(None, 36)	1332
dense_10	(Dense)	(None, 36)	1332
dense_11	(Dense)	(None, 36)	1332
dense_12	(Dense)	(None, 108)	3996
dense_13	(Dense)	(None, 108)	11772
dense_14	(Dense)	(None, 108)	11772
dense_15	(Dense)	(None, 108)	11772
dense_16	(Dense)	(None, 108)	11772
dense_17	(Dense)	(None, 36)	3924
dense_18	(Dense)	(None, 36)	1332
dense_19	(Dense)	(None, 36)	1332
dense_20	(Dense)	(None, 36)	1332
dense_21	(Dense)	(None, 36)	1332
dense_22	(Dense)	(None, 18)	666
dense_23	(Dense)	(None, 18)	342
dense_24	(Dense)	(None, 18)	342
dense_25	(Dense)	(None, 18)	342
dense_26	(Dense)	(None, 18)	342
dense_27	(Dense)	(None, 9)	171
dense_28	(Dense)	(None, 3)	30
=====			
Total			
params: 70,185			
Trainable			
params: 70,185			
Non-trainable			
params: 0			