Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrsthul für Datenverarbeitung

Applied Reinforcement Learning

Final Report – Group F

Game of Drones

Authors: B.Sc. Domenico Tomaselli
B.Sc. Michael Seegerer
B.Sc. Sayanta Roychowdhury

Supervisors: M.Sc. Martin Gottwald
Dr. Hao Shen

30.07.2020

Lehrsthul für Datenverarbeitung
Technische Universität München
Prof. Dr.-Ing. Klaus Diepold

List of Figures

List of Tables

Summary of Notation

Symbol	Description
W	Width of the environment
H	Height of the environment
(x_t, y_t)	Spatial coordinates at time t
(v_{x_t}, v_{y_t})	Linear velocity at time t
ϕ_t	Angle at time t
ω_t	Angular velocity at time t
d_{target}	Euclidean distance to target
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
s_t	Current state
s_{t+1}	Next state
u_t	Current action
u_{t+1}	Next action
r_t	Current reward
Q	Action–value function
α	Learning rate
γ	Discount factor
ϵ	Exploration policy parameter
u_y	Feedback y–position PID controller
u_ϕ	Feedback ϕ–position PID controller

Abstract

Reinforcement learning represents a branch of machine learning where the designed algorithm learns from the interaction with an environment; the learning process consists of mapping situations derived from the states of the environment to actions that the algorithm can carry out to maximize a numerical reward signal [1].

In contrast to other machine learning branches such as supervised and unsupervised learning, the algorithm, also denoted as the agent in the context of reinforcement learning, has no specific instructions on which actions to undertake, but instead it must discover how to respond to a given situation by first exploring each possibility and then isolating the one that returns the highest reward signal [1]. The formulation of a reinforcement learning problem must include in its simplest form three aspects. First, sensation, i.e. an agent must be able to sense the state of the environment. Second, action, i.e. an agent must be able to take actions that affect the state of the environment. Third, goal, i.e. the agent also must have a goal relating to the state of the environment that is characterized by yielding the most reward once it is reached [1].

One of the biggest challenges that arise when attempting to solve reinforcement learning problems is the trade-off between exploration and exploitation. The general objective of an agent is to isolate and perform those actions that ultimately maximize the reward signal, both short–term and long–term; to achieve such feature, the algorithm tracks based on a given sensation of the current environment which response that was already undertaken in previous iterations yields the most lucrative reward signal [1]. To effectively discover these productive actions, the agent is first subject to a phase of exploration, where given a certain state of the environment it undertakes a large variety of responses without focusing on what reward it's acquiring. In other words, an agent must exploit what it already knows in order to maximize the reward signal appropriately and to do so, it must first explore all possible responses to a given environment state till then to perform better action selections in the future [1]. Neither exploration nor exploitation can be exclusively pursued without failing at the given reinforcement learning task, there must reign a balance between the two aspects [1].

This report serves to provide a concise glimpse into the effort put forth to design the concept of a reinforcement learning environment from scratch, implement it as a Markov Decision Process such that the described reinforcement learning features are satisfied and architect an agent capable of absorbing the exploration/exploitation trade–off in the attempt to identify the most efficient solution. Specifically, the concept of the environment was deployed in the field on unmanned aerial vehicles (UAVs), where traditional engineering techniques, sufficiently advanced for the appropriate controlling of UAVs are often very challenging and the application of reinforcement learning could result in the design of an algorithm able to operate such aircraft without human supervision. The conceived environment accounts for the physics behind the lifting procedure of an UAV as well as maintaining its airborne status, while also incorporating common disturbances, e.g. the influence of wind. The agent implemented with the task of learning/solving the designed world must identify the actions that propel the drone to lift from a ground position, perform standard controlling procedures to maintain stability while airborne and guide the UAV towards a pre-defined target position whilst adjusting and counteracting the influence of the disturbances.

Contents

Chapter 1

Introduction

1.1 Motivation

While the use of UAV originated mostly for military purposes, their application towards other fields is rapidly growing and improvements in the technology of the aircraft are an active area of research. On the current market, drones equipped with high definition cameras can be easily purchased for a broad range of applications, nevertheless their control mechanism remains a challenging aspect if traditional engineering techniques are employed. The application of Reinforcement Learning for such task could potentially result in the successful design of an algorithm capable of learning how to operate the drone without human supervision and spare the effort currently invested in developing and implementing traditional controlling schemes.

There is a growing consensus in the aircraft industry to reduce the pollution caused by the aircraft; according to the European Aviation Environmental Report stipulated in 2019, between 1990 and 2016 the CO_2 emissions of all flights departing from EU28 and EFTA has increased from 88 to 171 million tonnes (+95 %). The application of Reinforcement Learning could have the potential to facilitate the alignment of UAVs to such consensus by expanding the design of the agent to optimize the fuel consumption while being airborne and navigating towards the desired target locations [2].

Last but not least, the vertical take off mechanism of an UAV from a ground position, which will be one of the central tasks required to be performed by our algorithm, remains an appealing topic from several perspectives; it is not merely a theory–based problem in terms of accounting for the physics that is involved, but it has substantial use–cases in practical applications and it is at first glance an operation characterized by a fairly restricted set of actions or rather a limited set of factors influence its mechanism.

1.2 Goals

The main objective of this work is the deployment of an environment capable of simulating the take off mechanism and the airborne status of an UAV and to investigate its controlling procedures according to the different aspects and factors influencing it.

The concept of the environment must resemble the quintessential physical and financial aspects that characterize a standard UAV airborne circumstance, including the lift generated from the different engines to elevate the drone vertically from its starting ground position and its counteracting gravitational force, the thrust/drag also produced by the different engines that steer the position of the drone horizontally and as well as potential disturbing factors such as the presence of four–directional wind forces with alternating strength or malfunctioning reception originating from the UAV's wireless controller. From a financial standpoint, our goal is to incorporate action patterns that influence the costs of operation such as the fuel/battery consumption resulting from operating the engines of the UAV and the time efficiency to reach the desired target position. From a more technical point of view, the concept consisted in constructing a two–dimensional environment simulating an airborne UAV in earth–like surroundings whilst satisfying the essential reinforcement learning features. The agent

implemented with the task of learning/solving the designed concept must be capable of performing a series of chores. First, it must identify those actions that allow for a take off mechanism to be performed in a safe fashion using separately controllable engines located on both sides of the drone and under the influence of the counteracting gravity pull. Second, once the take off procedure is complete, it must guide the drone towards a randomly specified target location using the least amount of fuel/battery capacity possible by activating the engines only when absolutely necessary. Third, to ensure the stability of the drone while airborne even under the influence of disturbances in the environment, it must execute actions that directly compensate the controlling procedure otherwise performed by a standard PID controller.

Regarding the prototype and the initial task to be performed, the goal was to limit the complexity of the environment to be solved by only attempting the take off procedure from a starting ground position and steering the UAV towards a defined target location placed vertically above it, i.e. the target's x–coordinate coincides to the drone's abscissa while the y–coordinate is randomly selected according to an uniform distribution. Once the initial task was successfully completed and the prototype validated the drone's physically correct behaviour with respect to all possible actions, the complexity of the agent's task was increased by randomly selecting x–coordinate of the target, including disturbances to update the concept into a more authentic, real–world adaptation and last but not least incorporate secondary control aspects such as optimizing the consumption of fuel/battery capacity or increasing the time efficiency.

1.3 Outline

The rest of the report is organized as follows. In Chapter 2, we provide a glimpse into the process of designing the concept of an OpenAI environment as a Markov Decision process, implementing it such as that the real–world physics and the Reinforcement Learning features are satisfied and ultimately constructing and training an agent capable of learning/solving it. In Chapter 3, we discuss what aspects from the original concept were successfully incorporated in the final version of the environment as well as insightful observations that we have drawn during its development/training. Finally, in Chapter 4 we conclude the report and consider potential improvements/adjustments to the structure of the environment for future outlooks. In Appendix A, we outline our effort to adapt the Reinforcement Learning algorithms applied on our own environment to worlds with seemingly similar structure, but of entirely different concepts.

Chapter 2

Concept and training

2.1 Development of the environment

The concept of the conceived environment originates from one of the OpenAI environments, i.e. the *LunarLander* [3]. With each milestone of the project and with the increased confidence we acquired in handling the simulation tool Box2D as well as the available OpenAI GYM packages, we distanced ourselves from the *LunarLander* and started implementing brand new features, such as wireless disturbances in the controller and wind forces. Ultimately, our objective was to develop the final version such that has no dependencies from the *LunarLander*.

2.1.1 State space

Considering the significant changes we implemented to the structure of the environment, the state space developed has substantial differences w.r.t. *LunarLander*.

Our initial idea consisted of discretizing the entire environment world into a two–dimensional grid; based on such concept the resulting state space was also arranged in a discrete domain. Considering that such a grid world is not the optimal solution to account for the physics and the features formulated in the concept and to better equip our agents in reaching the desired goals defined in Chapter 1.2, we have adjusted the state space to allow continuous dynamics in the prototype. The final version maintains the continuous dynamics approach of the prototype with certain tweaks and adjustments to improve the sensation quality that they provide.

The state space of the environment's final version is as follows:

Continous state space (At time instance t):

- Position (x_t, y_t): Distance vector between target and drone position
- Velocity (v_{x_t}, v_{y_t}) : Linear velocity of the drone in xy–coordinates
- Angle ϕ_t: Angle of the bottom platoon of the drone to the ground
- Angular velocity ω_t: Angular velocity of the drone.

Specifically, the state space contains the normalized distance coordinates between the drone and the target location in range [0,1], the angle of the drone in relation to the ground normalized in range [0,1] and prior to that shifted into $[0, 2\pi]$ from a $[-\pi, \pi]$ domain to avoid handling negative angle values. In addition to that, the state space also contains the normalized velocity vectors of the drone's linear and angular motion at a particular time instance.

The state space and the instance in which the environment can be considered to be solved are visualized in FIG. 2.1. To tackle the curse of dimensionality that arises from this six–dimensional state space and keeping in mind about making the task somewhat convenient for a Reinforcement Learning agent to identify an optimal policy, we

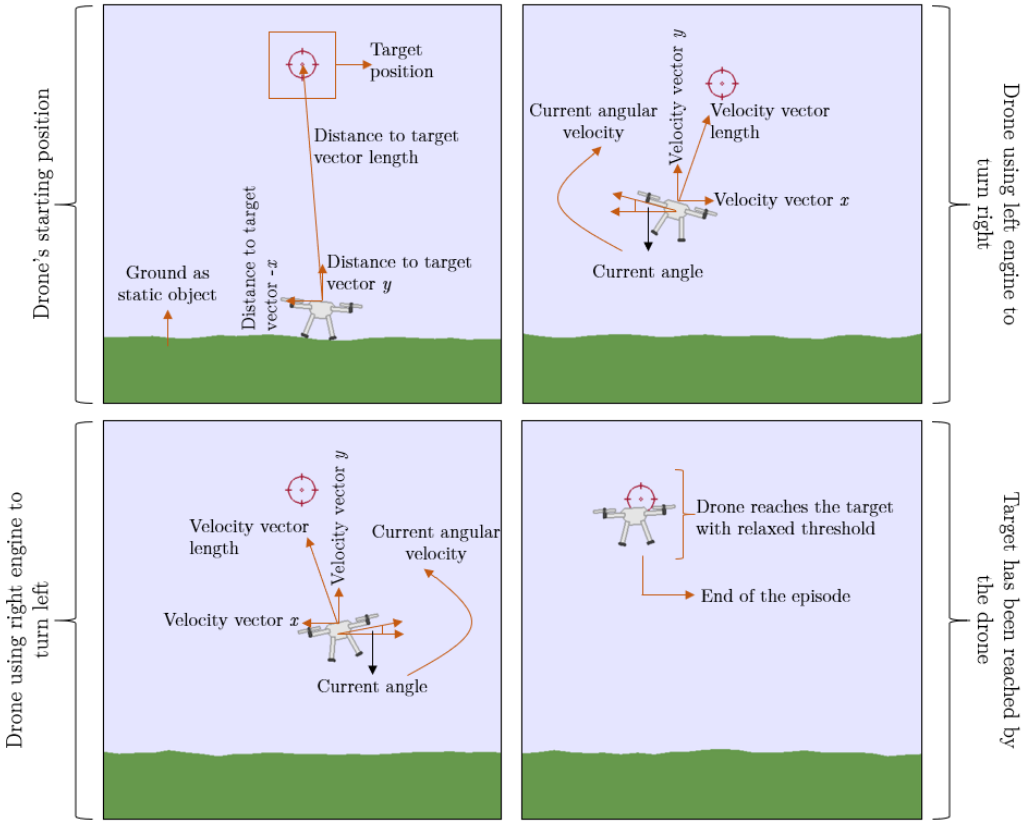


FIG. 2.1: Visualization of the state space with four different instances of the environment.

introduced three parameters utilizing the available information from the state space. Considering W and H as the width and height of our rendering window, they can be expressed as follows:

(i) *Euclidean distance to target:* The normalized l_2–distance between the drone's surface and the target position at time instance t,

$$d_{\text{target}} = \sqrt{\left(\frac{x_t^2 + y_t^2}{W^2 + H^2}\right)} \tag{2.1}$$

(ii) *Effective drone angle to target:* The most efficient and direct route to the target results from aligning the linear velocity vector to the direction of the distance vector. To translate such observation into a measurable parameter, we introduced a difference measure between the current direction of the velocity vector and the distance vector.

$$\theta_{diff} = \frac{\arctan 2\left(\frac{y_t}{x_t}\right) - \arctan 2\left(\frac{v_{y_t}}{v_{x_t}}\right)}{2\pi} \tag{2.2}$$

where arctan2 is an arctangent function that chooses the quadrant correctly based on the input values.

(iii) *Length of the velocity vector:* The resulting velocity of the drone estimated according to the velocity components v_x and v_y at time t.

$$v_{\text{length}} = \sqrt{\left(\frac{v_{x_t}^2 + v_{y_t}^2}{W^2 + H^2}\right)}, \tag{2.3}$$

where the velocity vector length is normalized between [-1,1] and the components v_x and v_y are appropriately scaled to ensure the v_{length} does not exceed a defined velocity threshold.

A graphical explanation of the reduced state space representation is shown in FIG. 2.2.

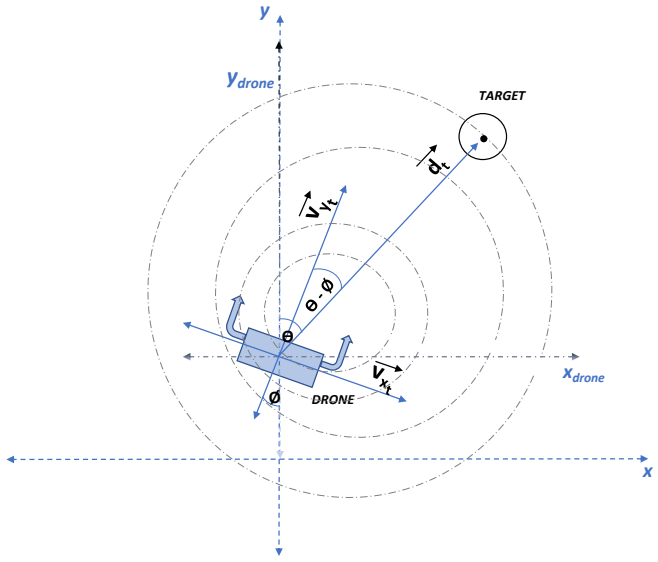


FIG. 2.2: Modified state space representation; ϕ_t is the drone angle to the ground, θ is the drone head angle w.r.t. target, $\vec{d_t}$ is the distance vector and (v_{x_t}, v_{y_t}) denoting the velocity components along xy coordinates

2.1.2 Action space

Regarding the action space, we have four possible actions related to the drone's engines:

Action space prototype:

- $u_t = 0$ if both left and right engines deactivated
- $u_t = 1$ if right engine activated/left engine deactivated
- $u_t = 2$ if left engine activated/right engine deactivated
- $u_t = 3$ if both left and right engines activated

Considering the downsampling of the environment from three to two spatial dimensions, the number of engines employed for the drone's navigation is reduced from the standard four engines to a two engines configuration. Based on this design, the agent/algorithm will be prompted to undertake one of four different combinations of actions at each time step t in the attempt to complete the assigned tasks: (i) activate both right and left engine to enforce a greater thrust than the counteracting gravitational force and induce the ascending of the aircraft, (ii) activate either the right or the left engine to enforce a greater thrust only on the side of the drone where the engine is turned on or (iii) shut down both right and left engine generating not enough thrust to withstand the counteracting gravitational force and causing the drone's descending.

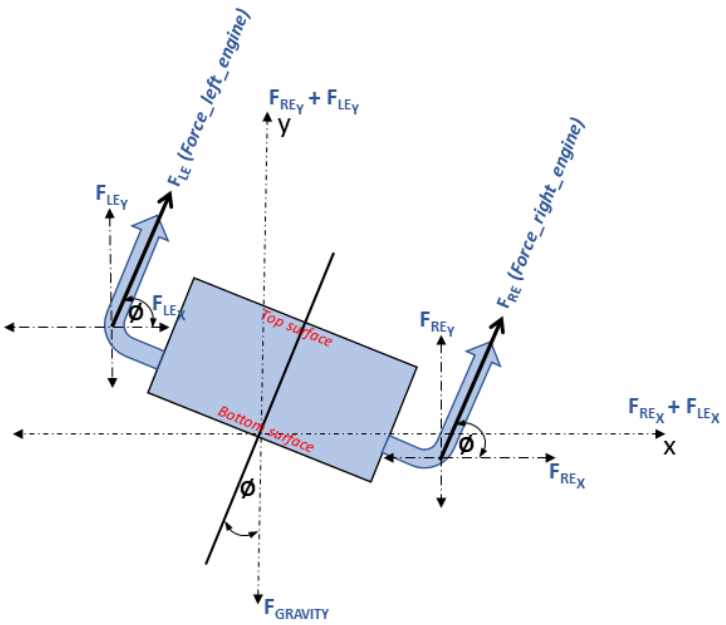


FIG. 2.3: Visualization of the force body diagram exhibiting the direction of all forces acting upon the drone when airborne in the designed environment.

2.1.3 State transitions

Following the force distribution of the drone engines and gravity as depicted in FIG. 2.3, we have arranged the continuous dynamics for drone state transition according to:

State transitions:

- Left and right engine are shut down ($u_t = 0$) : Gravitational pull forces the drone to descend according to Box2D world gravity initialization.
- Left engine shut down/right engine activated ($u_t = 1$): Right engine running at full capacity propels drone's right side upwards along the ordinate according to the thrust generated from $F_{\mathrm{RE}_Y} - F_{\mathrm{GRAVITY}}$ and sideways along the abscissa according to the thrust generated from F_{RE_X}.
- Left engine activated/right engine shut down ($u_t = 2$): Left engine running at full capacity propels drone's left side upwards along the ordinate according to the thrust generated from $F_{\mathrm{LE}_Y} - F_{\mathrm{GRAVITY}}$ and sideways along the abscissa according to the thrust F_{LE_X}.
- Left and right engines are activated ($u_t = 3$): Both left and right engines running at full capacity, the resultant thrust generated along the ordinate $F_{\mathrm{RE}_Y} + F_{\mathrm{LE}_Y} - F_{\mathrm{GRAVITY}}$) propels the ascending of the drone while the thrust $F_{\mathrm{RE}_X} + F_{\mathrm{LE}_X}$ generates the resulting movement along the abscissa.
- The angle ϕ_{t+1} occurring from the corresponding motion of the drone is estimated according to the Box2D world step function.
- Turmoil caused by climatic agents: wind can affect the drone's current route by increasing/decreasing its current velocity as well as producing undesired horizontal shifts of its current position along.

2.1.4 Reward function

The reward function of the prototype consisted of a primary and secondary agent where each agent was assigned to perform different tasks such as reducing the distance to the target, maintaining stability while being airborne, etc. Due to time restrictions and unexpected complications resulting from the environment, we reduced the complexity of the reward function and merged the functionality of the two agents into a single one. The final reward function is as follows:

Reward function:

$$r(s_t) = r(x_t, y_t, \phi_t) = \begin{cases} -10 \cdot d_{\text{target}} & \\ +250, & \text{if } d_{\text{target}} < (1/\sqrt{W^2 + H^2}) \\ -10, & \text{if } \phi_t < \frac{0.5}{2\pi} \vee \phi_t > \frac{1.5}{2\pi} \\ -10, & \text{if } x_t > 1 \vee x_t < 0 \vee y_t > 1 \end{cases} \tag{2.4}$$

where d_{target} is the normalized euclidean distance between the upper surface of the drone and the target position according to (2.1), the scaling factor $\sqrt{W^2 + H^2}$ is included to normalize the threshold determining whether the drone has reached the target and the scaling factor 2π is applied to normalize the threshold for determining whether the angle of the drone is exposed beyond physically possible boundaries.

The agent will receive an overall reward of 250 if it successfully guides the drone in proximity of the target, namely $1/\sqrt{W^2 + H^2}$. The main punishment of the agent is determined according to the current distance from the target, where the Euclidean distance is the metric used to estimate $(-10 \cdot d_{\text{target}})$. The purpose behind such continuous penalty is to push the agent to reduce the distance from the target swiftly and in efficient fashion. A second penalty results from the current angle of the drone. To guarantee stability while being airborne, the UAV must maintain small angles; once the given threshold is exceeded, a penalty of 10 will be enforced to punish resulting unstable behaviours. A last obvious punishment must be introduced in case the drone exits the environment; considering the world in which the drone operates has no restrictions in terms of the sky, an instance in which the UAV goes beyond the limits can take place and must be avoided.

2.1.5 Additional environment features

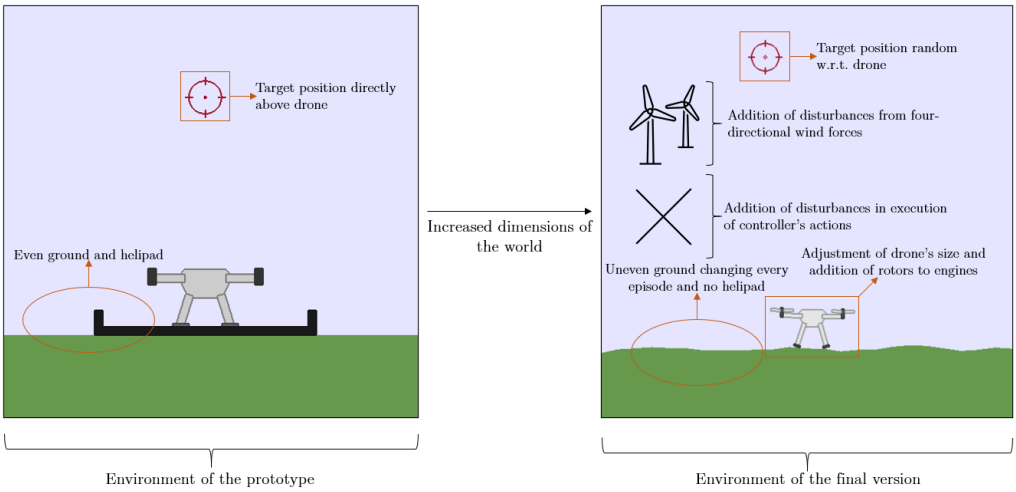


FIG. 2.4: Visualization of the updates applied to the rendering of the prototype as well as of the addition of supplementary features to reinforce the plausibility of the environment.

In the final version of the environment a number of improvements were introduced

to its rendering as well as features to provide a more sophisticated and realistic representation of the original concept. As visualized in FIG. 2.4, the dimensions of the UAV were adjusted to a more realistic size w.r.t. to the overall dimensionality of the world and its engines were upgraded from a graphical point of view with the addition of rotors. The overall size of the environment was better conformed and the height was slightly incremented to add more room to operate for the drone considering that in the final version aspects such as thrust and drag are introduced.

The supplementary features introduced in the final version of the environment are as follows.

(i) *Uneven ground:* The terrain of the prototype was perfectly even and a helipad highlighted the starting position of the UAV. Such scenario was appropriately edited in the final version of the environment by abandoning the concept of a helipad and introducting terrain irregularities to establish a more realistic representation of the drone's soundings.

(ii) *Random starting/target location:* In the prototype the task to be performed by the UAV was elementary, considering that the x–coordinate of the target position was purposely aligned to the x–coordinate of the drone. Such scenario does not account for the thrust and drag of the UAV since only the lift is required to reach the target. To address this initially desired deficiency, both the drone's starting position as well as the target position are sampled at the start of every episode from a set of five random spatial coordinates.

(iii) *Malfunctioning in wireless communication:* UAVs are often controlled remotely and there are precedents for not executing the instructed/transmitted behaviour. To incorporate this deficiency in the final version of our environment, we re–formulated the process of taking actions to a discrete probability distribution $P(U)$ that we defined as follows:

$$\mathrm{P}(U = u_{t,i}) = 0.9, \quad \mathrm{P}(U = \text{random}) = 0.1, \quad \sum \mathrm{P}(U) = 1, \tag{2.5}$$

where $0 < \mathrm{P}(U) < 1$ and U represents the random variable determining which action will be ultimately carried out, either the desired action (probability of 0.9) or a random action (probability of 0.1).

(iv) *Random wind forces.* During take–off/flying procedures an UAV is constantly exposed to different wind forces that could affect its trajectory and part of the controller's task is to account for such disturbances and absorb them such that the final task of reaching the desired position is unaffected. To incorporate this atmospheric disturbance in the final version of the environment the model was further expanded with four–directional random wind forces F_w as follows:

$$F_w = \begin{cases} \text{from the bottom,} & \text{if } 0 \geq P(F_w) \leq 0.25 \\ \text{from the top,} & \text{if } 0.25 \geq P(F_w) \leq 0.5 \\ \text{from the left,} & \text{if } 0.5 \geq P(F_w) \leq 0.75 \\ \text{from the right,} & \text{if } 0.75 \geq P(F_w) \leq 1.0 \end{cases} \tag{2.6}$$

In (2.6), F_w randomly changes direction and amount of force being applied at each episode. If F_w is from the bottom, the linear velocity of the drone is incremented; if F_w from the top, the linear velocity of the drone is reduced; if F_w is from the right, the drone is constantly shifted towards the left and must learn how to counterbalance the disturbance; if F_w from the left, the drone is constantly shifted towards the right and it must learn how to counterbalance the disturbance.

2.1.6 Testing Drone Environment with an keyboard agent

In the prototype, we demonstrated how the UAV was capable of performing the take–off procedure and reach a target location placed directly above it. After increasing

the complexity of the task at hand, the environment was immediately subject to complications needed to be addressed appropriately. To that extent, we developed a keyboard agent to perform debugging as well as investigate the dynamics and the reward function with more details.

As outlined in the action space, the four possible actions that the drone can take to navigate in the environment correspond to the integers within the interval [0,3]. The keyboard agent exploits this simple design to support the maneuvering of the UAV using the keyboard numbers $[0, 1, 2, 3]$. Furthermore, one can pause the environment by pressing the space–bar in case a closer look at a specific time instance is required. Overall, the keyboard agent was a great tool to quickly familiarize ourselves with environment and uncover bugs/inaccuracies in its design. Specifically, among the deficiencies that the prototype environment contained, a few were identified and properly addressed with help of the keyboard agent.

(i) We noticed how the density of the UAV was initially set too low and this resulted in the issue of the drone not being subject to the gravitational force.

(ii) We observed how the impulse resulting from engine usage was set too high; this propelled the issue of excessive angular velocity/rotations of the drone.

(iii) Considering that the keyboard agent allowed us to see the reward function at each time instance, we used it extensively to ensure that the wrong behaviours were punished accordingly.

(iv) By testing the environment with random policies, we quickly realized that it was mandatory to insert a number of conditions to ensure that the drone would not be allowed to perform certain physically incorrect maneuvers. Among these conditions, one was set to avoid that the drone would remain on the ground, but as it turned out it was wrongly formulated and constantly caused the breaking of the environment. Such bug was uncovered with the keyboard agent and appropriately addressed.

(v) To provide the same state bounds for all states we performed normalization; regarding the velocity/angular velocity the keyboard agent was extremely helpful to analyze and determine what could be the threshold to set.

2.2 Application of Reinforcement Learning

We designed our environment as a Markov Decision Process so that traditional Reinforcement Learning algorithms that do not require the use of Neural Networks can be applied to solve it.

Considering that to better account for the behaviour of the drone we exploited continuous state dynamics, solving the environment using traditional Reinforcement Learning methods without the application of Deep Learning would most likely be an unfeasible task, we transformed the scaled three–dimensional state space into a discrete domain. To perform a proper dicretization, we defined a three–dimensional non-uniform grid, where each dimension refers to the number of bins N_d of respectively *distance to target*, *effective drone angle* and *resultant drone velocity* respectively.

Specifically, the continuous state space is discretized according to the following steps:

State space discretization:
The bin index i is estimated according to:

$$i = \begin{cases} 0, & \text{if } s_t = a \\ N_d - 1, & \text{if } s_t = b \\ \left\lfloor \frac{(s_t - a) \cdot N_d)}{(b-a)} \right\rfloor, & \text{if } a < s_t < b \end{cases} \tag{2.7}$$

where the continuous state space variables are $s_t \in [a, b]$ and the index $i \in \{0, N_d - 1\}$.

2.2.1 Monte Carlo

The term "Monte Carlo" is often used more broadly for any estimation method whose operation involves a significant random component. Monte Carlo (MC) methods applied to Reinforcement Learning tasks require only experience – sample sequences of states, actions, and rewards from actual or simulated interaction with an environment [1]. Contrary to Dynamic Programming (DP), MC–based algorithms do not required to know the full probability distributions of all possible state transitions in the model; the focus is instead placed on solving a Reinforcement Learning problem based on episodic average returns. MC–based algorithms are generally of two typed: *first–visit MC algorithm* and *every–visit MC algorithms*. The *first–visit* approach estimates the value function $v_\pi(s)$, where π is the optimal policy, according to the average of the returns resulting from the first visits to the state s_t, whereas the *every–visit* approach averages the returns resulting from all the visits to the state s_t [1].

To solve the environment, we implemented the *first-visit MC algorithm* using a ϵ–greedy policy according to (2.9) as follows:

First-visit Monte Carlo control algorithm:

Initialize $Q(s_t, u_t) = -10000, \quad \forall s_t \in \mathcal{S}\ , \quad \forall u_t \in \mathcal{A}$
Initialize returns $G(s_t, u_t) = 0$

Repeat for each episode, $i = 0, 1, 2, ..., E-1$ (E is total number of episodes):
  Initialize $s_0 \in \mathcal{S}$ by resetting environment
  Discretize s_t
  Repeat for each step of the episode, $t = 0, 1, 2, .., T-1$ (T is maximum step limit):
    Choose action u_t from s_t derived from current Q according to ϵ–greedy policy
    Take action u_t, accumulate reward $r_t(s_t, u_t)$ and next state s_{t+1}
    $s_t \leftarrow s_{t+1}$
    $u_t \leftarrow u_{t+1}$
  until s_t is terminal

  Sequence of states, actions and rewards $s_0, u_0, r_0, ..., s_{T-1}, u_{T-1}, r_{T-1}$ are generated
  Repeat for each unique (s_t, u_t) pair over the episode i:
    Enumerate through the episode to find first occurrence index j of (s_t, u_t)
    Repeat till end of episode $k = j, j+1, ..., T-1$:
      $G(s_t, u_t) \leftarrow G(s_t, u_t) + r_k \cdot \gamma^k$
      $Q(s_t, u_t) \leftarrow average(G(s_t, u_t))$

Since the environment yields positive reward only when the agent reaches the target position, the Q table must be initialized with a negative value instead of with zeroes. The ϵ–greedy policy when determining the action to be taken computes the maximum Q value for a given state s_t. If the Q table is initialized at zeroes, it could be case that by taking the maximum an unexplored action is selected, which is not desirable.

Since one of the keys for the convergence of this algorithm is the amount of exploration of the state space, ϵ as the exploration parameter in the ϵ–greedy policy had to be adequately tuned to achieve sufficient balance in the trade–off between exploration and exploitation. Also, since long–term rewards are essential for reaching the target position in our environment, the discount–factor γ was also considered in the hyper–parameter tuning. The final tuning of the hyper–parameters was determined according to the amount of states that were explored and the number of times the drone is sucessfully guided to the target.

Hyperparamater	**Value**
Discount factor γ	0.9
Exploration parameter ϵ	0.2
Discretization of the state space	(20,10,5)

TABLE 2.1: Final tuning of the main hyper-parameters for Monte Carlo control.

2.2.2 Q–Learning

Q–learning is an off–policy TD algorithm that originated from the idea of Watkins in [4]. In Q–learning, the learned action–value function Q directly approximates the optimal action–value function $Q*$ independent of the policy being followed. Specifically, the policy does still play a role in a sense that it determines the which state–actions pairs are being visited, but the purpose of converging correctly it is only required that all state–actions pairs are continuously updated. The Q–learning algorithms is as follows:

Q–learning algorithm:

Initialize $Q(s_t, a_t) = -1000, \quad \forall s_t \in \mathcal{S}, \quad \forall u_t \in \mathcal{A}$
Repeat for each episode: Get decaying ϵ according to ϵ–decay
  Initialize s_t by resetting environment
  Discretize s_t
  Repeat for each step of the episode:
    Take action u_t, observe $r(s_t, u_t)$ and s_{t+1} according to ϵ–greedy
    $Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \cdot [r(s_t, u_t) + \gamma \cdot \max_{u_t} Q(s_{t+1}, u_t) - Q(s_t, u_t)]$
    $s_t \leftarrow s_{t+1}$
  until s_t is terminal

While the concept of the algorithm is easy, when exposed to a complex environment such as the one we have designed, a high level of hyper–parameters tuning must be performed to find the optimal combination and extra features must be introduced to ensure that a certain balance between exploration and exploitation reigns.

Additional features:

(i) ϵ–*decay:* The function is an essential element to guarantee that enough exploration is being performed by the agent. The ϵ value determines in ϵ–greedy the policy that the agent will follow and ensuring that ϵ decays during the course of the training allows the agent to initially focus on the exploration of the environment and only after a number of episodes determined according to the hyper–parameter ϵ_{decay} shift to the exploitation. ϵ–*decay* is implemented as follows:

$$\epsilon = \epsilon_{\text{MIN}} + [(\epsilon_{\text{MAX}} - \epsilon_{\text{MIN}}) \cdot \exp(-\epsilon_{\text{decay}} \cdot \text{episode})], \tag{2.8}$$

where $\epsilon_{\text{MIN}} = 0.01$ and $\epsilon_{\text{MAX}} = 1.0$, ϵ_{decay} is the hyper–parameter that determines how fast the ϵ should decay and for *Q–learning* it was set at 0.0001.

(ii) ϵ–*greedy policy:* ϵ–*greedy* is the policy that the *Q–learning* agent follows. It takes as input the ϵ value computed in ϵ–*decay rate* and based on that it determines whether the action to be pursued by the agent is random for exploration purposes or the action yielding the highest state–action value for exploitation purposes.

$$u_t = \begin{cases} \arg\max\{Q(s_t)\}, & \text{if } rand() > \epsilon \\ \text{random}, & \text{else} \end{cases} \tag{2.9}$$

where $rand()$ is a uniformly distributed random scalar from the interval (0,1) and random denotes an arbitrary element of the action space.

Regarding the hyper–parameters, the final tuning is according to TABLE 2.2. A learning rate α set too lows does not yield any learning, while if α is set too large, the agent learns too fast overshooting the global minimum. The discount factor γ is set as high as possible to emphasize the long–term reward instead of the short–term. The ϵ–decay rate is set very low to ensure required exploration of the environment; the larger the dimensions of the state–action function, the lower is the ϵ–decay rate must be to guarantee enough exploration. The number of episodes is the trade–off between running time and ensuring the agent has enough episodes to maintain the required balance between exploration and exploitation.

2.2.3 SARSA

SARSA is also a TD based approach, but contrary to *Q*–learning it belongs to the class of *on–policy* learning algorithms. The idea behind SARSA originates from Rummery in [5] and was introduced as *Modified Connectionist Q-Learning (MCQ-L)*. Its functionality is similar to a *Q*–learning procedure which are described in more details in 2.2.2. The major difference between the two methods is that contrary to *off-policy Q*–learning, the

Hyperparamater	**Value**
Learning rate α	0.1
Discount factor γ	0.99
ϵ–decay rate	0.0001
Number of episodes	200000
Discretization of the state space	(20,10,5)

TABLE 2.2: Final tuning of the main hyperparameters for Q–learning.

update of the action–value function does not depend on the maximum possible reward of the next state, instead SARSA considers the observed reward that it would receive by following the derived policy at the next time instance.

To ensure a better comparability with Q–learning, the policy used in SARSA was also ϵ–greedy that is described in more detail in the previous section. In SARSA, ϵ–decay according to (i) is also used to ensure that enough exploration happens at the start of the learning phase.

The learning principle of SARSA can summarized with equation 2.10, here the action taken at time t are notated as u_t and the state at time t as s_t.

$$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \cdot [r(s_t, u_t) + \gamma \cdot Q(s_{t+1}, u_{t+1}) - Q(s_t, u_t)] \tag{2.10}$$

A complete pseudo-code summary of the used SARSA algorithms is as folllows:

SARSA algorithm:

Initialize $Q(s_t, a_t) = -1000, \quad \forall s_t \in \mathcal{S}, \quad \forall u_t \in \mathcal{A}$
Repeat for each episode:
    Initialize s_t by resetting environment
    Discretize s_t
    Get decaying ϵ
    Choose action u_{t+1} from s_t derived from Q according to ϵ–greedy
    Repeat for each step of the episode:
        Take action u_t, observe $r(s_t, u_t)$ and s_{t+1}
        Choose action u_t from s_{t+1} derived from Q according to ϵ–greedy
        $Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha \cdot [r(s_t, u_t) + \gamma \cdot Q(s_{t+1}, u_{t+1}) - Q(s_t, u_t)]$
        $s_t \leftarrow s_{t+1}$
        $u_t \leftarrow u_{t+1}$
    until s_t is terminal

The final tuning of the hyper-parameters is according to TABLE 2.3. The ϵ–decay rate is designed such that the agent focuses in the first quarter of the learning period on the exploration aspect. The total number of episodes is set only at 10000 due to large running time required by the algorithm.

Hyperparamater	**Value**
Learning rate α	0.3
Discount factor γ	0.95
ϵ–decay rate	0.00015
Number of episodes	10000
Discretization of the state space	(20,10,5)

TABLE 2.3: Final tuning of the main hyper-parameters for SARSA.

2.3 Evaluation

2.3.1 Monte Carlo

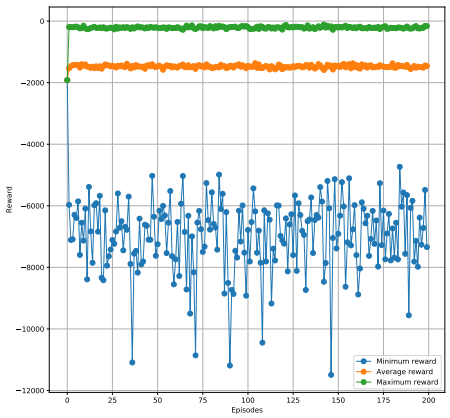
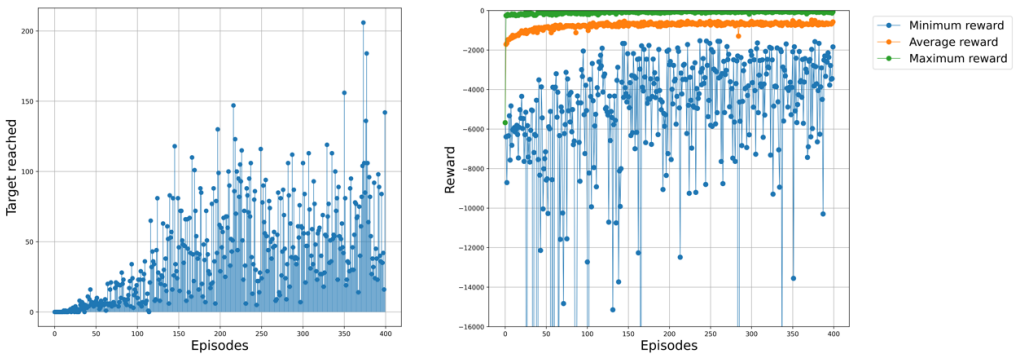


FIG. 2.5: Minimum, average and maximum reward distribution of Monte Carlo training with intervals of 500 episodes.

The MC agent was trained over 200000 episodes with maximum 1000 steps per episode for a wide range of ϵ and γ combinations.

Despite the extensive training, the agent was not able to perform any learning. The minimum, average and maximum reward distribution are visualized in FIG. 2.5; the trajectory of the distribution clearly indicates how there is no ascending.

2.3.2 Q–Learning



(a) (b)

FIG. 2.6: (a) number of times the agent is able to navigate the UAV to the desired target location with intervals of 500 episodes. (b) Minimum, average and maximum reward of the training with intervals of 500 episodes. The y–axis is purposely limited to discard minimum rewards beyond a certain threshold that can be considered as outliers.

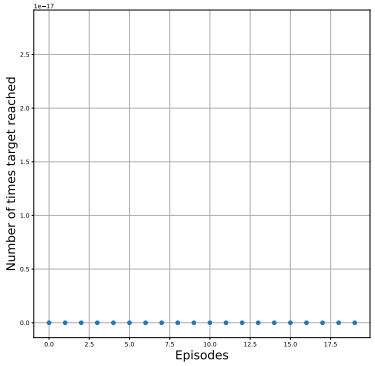
As FIG. 2.6 shows, after an initial exploration phase dictated by the ϵ–decay/ϵ–greedy policy, the agent starts to learn and it makes certain progress. Nevertheless, after approximately 100000 episodes no progress is made in terms of learning. This pattern is also visible, when the resulting Q table is evaluated; the drone is navigated towards the target location and approximately 15 % of the times it manages to reach it, but the remaining 85 % it fails to do so.

We have several hypotheses as to why the agent stops learning after a certain number of episodes, e.g. the reduced state space does not incorporate enough

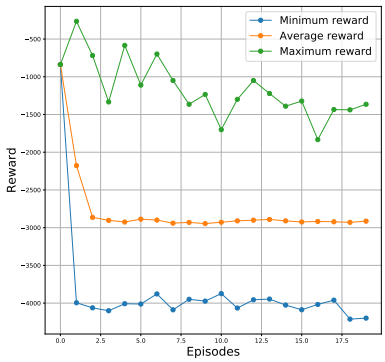
information about the environment, the discretization step results in a loss of information from which the agent cannot recover, etc. Regardless of the reason, the agent is not able to fully comprehend the environment and further developments would be required.

The results provided in FIG. 2.6 are obtained with the lowest complexity of the environment, i.e. no controller disturbances and additional wind forces are applied; an attempt was made to train the environment with the addition of the complexities, but unfortunately the training did not converge.

2.3.3 SARSA



(a) Number of times the agent is able to navigate the UAV to the desired target location with intervals of 500 episodes.



(b) Illustration of the accumulated rewards differed between the minimum, average and maximum reward of the training with intervals of 500 episodes.

FIG. 2.7: Training results of the SARSA agent. The learning was carried out over 10000 episodes in total. The agent learned here as an strategy to stay on ground.

The FIG. 2.7 illustrates the training results with our implemented SARSA approach (more detailed explained in the in section 2.2.3). Due to time and computational complexity, SARSA is only trained for 10000 episodes. To ensure enough exploration the decay rate was set at 0.00015.

The SARSA agent was also unable to perform any proper learning, instead reducing the distance to the target, a different pattern was learned, i.e. that it is more beneficial to stay on the ground and apply no engines usage. After approximately 2500 episodes, around the time where the exploration starts to significantly decrease and the taken action exploits more on the learned policy, the learning curve starts to fail. The reason for this lies probably in the reward function since the principle of the used SARSA approach was successfully tested with the Flappy Bird environment, as shown in the appendix at A.1. Another possibility could be that the agent is stuck in local minimum and therefore needs more episode to train on and a slower decreasing ϵ.

The results provided in FIG. 2.7 are obtained with the lowest complexity in the environment, i.e. no controller disturbances and additional wind forces are applied here.

2.4 PID controller

In the prototype we proposed an agent framework consisting of two agents working parallel to solve the task of the environment, i.e. navigating the drone to a pre–defined target location in efficient fashion. Specifically, the primary agent was designed to perform the optimal path finding, i.e. identify those actions that decrease the distance to the target. The secondary agent on the contrary was conceived to exclusively stabilize the UAV while airborne. Especially with the introduction of the additional

features to the environment described in Section 2.1.5, having an agent whose focus is placed only on minimizing the distance to goal is problematic as standard controlling procedures to maintain the stability of the drone would not be appropriately considered.

As the secondary agent, we proposed an approach that focuses on optimizing the hyper–parameters of a classical PID controller so that the primary agent can be fully committed at resolving the the task of minimizing the distance to the target.

To ensure that the secondary agent learns the correct behaviour, we designed the reward function as follows:

Reward function secondary agent:

$$r(s_t, u_{t,i}) = r(x_t, y_t, \phi_t, u_{t,i}) = \begin{cases} -1, & \forall t \in T \\ -10 \cdot v_{\text{drone}}, & \text{with } v_{\text{drone}} = \|(v_{x_t}, v_{y_t})\| \\ -10 \cdot v_{\phi}, & \end{cases} \tag{2.11}$$

where T is the total number of time steps, v_{drone} is euclidean norm of the linear velocity of the drone and v_ϕ the current angular velocity of the drone.

A classical PID controller is defined according to (2.12). Its hyper–parameters K_p, K_i, K_d are tuned by the secondary agent to preserve the stability of the drone, as investigated in [6].

PID Controller:

$$\mathbf{G}(s) = \mathrm{K_p} + \mathrm{K_i} \cdot \frac{1}{s} + \mathrm{K_d} \cdot s, \tag{2.12}$$

where $\mathrm{K_P}$ is the proportional term of the PID controller that provides an overall control–action relation to the resulting error signal. In our environment it relates for instance to the current total distance between the drone and a target location, where as the target location we defined the state drone location when we activated PID controlling. $\mathrm{K_I}$ is the integral gain whose task is the reduction of the steady–state errors through low-frequency compensation by an integrator and the derivative gain $\mathrm{K_D}$ improves the transient response through high–frequencies; in the context of the drone, it means that we are trying to minimize its velocity.

An insightful overview regarding the influences of the PID hyper-parameters is illustrated in Figure 2.8. The analysis was determined by Kiam Heong An and Gregory Chong in [7].

TABLE I
EFFECTS OF INDEPENDENT P, I, AND D TUNING

Closed-Loop Response	Rise Time	Overshoot	Settling Time	Steady-State Error	Stability
Increasing K_P	Decrease	Increase	Small Increase	Decrease	Degrade
Increasing K_I	Small Decrease	Increase	Increase	Large Decrease	Degrade
Increasing K_D	Small Decrease	Decrease	Decrease	Minor Change	Improve

FIG. 2.8: Overview of how the parameter tuning influences the system dynamics in [7]

Regarding the drone dynamics of the environment, compensating only the error resulting from detaching the PID starting location is not sufficient. The angle of the drone ϕ_t must also be appropriately controlled to ensure the overall stability of the

aircraft.

To achieve this controlling procedure, we introduced a second PID controller with the same structure as the first, only here the focus is placed on stabilizing the angle ϕ_t around 0°.

To decide which of the two PIDs has the priority over the other, we established a criterion according to (2.13) and (2.14) to identify and prioritize which instance (position or angle) requires the most urgent adjustment. In the case where both criteria are below a certain threshold, no action will be applied to drone, as stability is already in place.

This results in the following design:

PID controller design:

$$\xi_y = \Delta_{y_t} \cdot \mathrm{K_p} + \left(\int_{t=0}^{T} \Delta_{y_t} \mathrm{dt} \right) \cdot \mathrm{K_i} + \Delta_{v_{y_t}} \cdot \mathrm{K_d} \tag{2.13}$$

$$\xi_\phi = \Delta_{\phi_t} \cdot \mathrm{K_p} + \left(\int_{t=0}^{T} \Delta_{\phi_t} \mathrm{dt} \right) \cdot \mathrm{K_i} + \Delta_{\omega_t} \cdot \mathrm{K_d} \tag{2.14}$$

where Δ_{y_t} is the position error of the drone at time t, Δ_{ϕ_t} is the angle error at time t, $\Delta_{v_{y_t}}$ is the error of the drone's linear velocity along the y–direction and Δ_{ω_t} is the error of the drone's angular velocity.

Resulting Actions:

$$u_t = \begin{cases} 3, & \text{if } \xi_y > u_\phi \ \ \& \ \ \xi_y > \text{threshold} \\ 2, & \text{if } \xi_y < u_\phi \ \ \& \ \ \xi_\phi < \text{threshold} \\ 1, & \text{if } \xi_y < u_\phi \ \ \& \ \ \xi_\phi > \text{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

By implementing the defined PID controller using a rough estimation of the hyper–parameter, i.e. $\mathrm{K_p} = 0.5, \mathrm{K_i} = 0.5, \mathrm{K_d} = 0.5$ we already observed the desired stabilizing behaviour.

The introduction of the additional features to the environment as described in Section 2.1.5 exposed a significant drawback of our PID implementation. The PID controller is capable of rectifying variations along the y–coordinate and of the drone's angle ϕ_t, but in the case thrust and drag are introduced in the physics of the environment and the UAV receives velocity component also along x–coordinate, the PID controller is not cable of correcting it.

As a result of that, the drone started to drift parallel to x–coordinate of the environment traversing out of the window and since we were not able to isolate the source of this deficiency in the structure of the PID, we did not pursue the idea of having two separate agents any further.

Chapter 3

Discussion

This report serves as a general overview regarding the implementation of an OpenAI–like environment. We started with a project proposal, then proceeded to deploy a prototype and ultimately upgraded it into the final version by introducing additional features to make it more realistic. Specifically, we based the concept of the environment on UAVs, considering the increased application of the aircraft on a broad spectrum of practical fields. From a more technical point of view, the controlling procedures of UAVs using traditional engineering techniques represent a significant challenge and the objective of our concept was to evaluate whether Reinforcement Learning could be suitable for sparing the effort currently invested in deploying such procedures and control the aircraft without any human supervision.

After completing the general concept of the environment, we developed a first prototype to implement its initial rendering and started testing certain basic features, i.e. the take–off procedure, the counteracting gravitational force. Once the prototype was successfully deployed, we expanded the world by introducing additional features with the objective of improving the overall quality and authenticity of the environment. Furthermore, we included a number of real–world disturbances to increase the complexity of the environment and make the controlling procedures required to be performed by the agent more challenging.

The first upgrade deployed on the prototype was to improve the overall rendering. The graphical visualization of the prototype was elementary and included certain features that do not necessarily translate into real–world instances, e.g. a perfectly even terrain and a helipad corresponding to the starting position of the drone. The upgrade was twofold. First, the use of a helipad was discarded and the terrain from where the drone takes–off was adjusted into an uneven and more realistic variant. Second, the rendering of the drone was updated by reducing its dimensions whose ratio was too large w.r.t. the overall dimensions of the world and by introducing certain details aiming to improve the overall design of the UAV, e.g. adjusting the dimensions of the feet and adding two rotors to both engines.

The objective of the prototype consisted only performing the take–off procedure in a safe fashion and reach a target location placed directly above it, i.e. with the same x–coordinates. While the task is intrinsically elementary, it provided us with the necessary insights to determine if the concept was coherent and if the physics was correctly accounted for. This scenario has a limited number of factors influencing it, e.g. it does not account for the thrust and drag of the UAV since only the lift is required to reach the target. To address this initially desired deficiency, both the drone's starting position as well as the target position were sampled at the start of every episode from a set of five random spatial coordinates.

The described upgrade of the rendering and the randomly selected starting and target positions represent the starting point for the training of the agents. The complexity of the environment is still limited and it does not translate to real–world examples, but it represents a significant improvement w.r.t. the prototype. Our next objective was to introduce additional features such that the environment develops into a more realistic version. One of the things we noticed when looking into the standard controlling

procedures of UAVs, is that they are remotely controlled. Whether it is under human supervision or using AI, the corresponding manoeuvres are provided to the drone via wireless connection. To that extent, it can often occur while piloting UAVs that the transmission encounters a certain degree of disturbance and that the drone does not always manoeuvre as desired. This complication in the controlling procedure is the first level of complexity in our environment.

Another aspect that influences the proper navigation of UAVs are atmospheric conditions, e.g. wind. In real–world instances, drones are often exposed to wind forces in every phase of the flying procedures, i.e from when they take–off until they have performed the desired task. To account for such atmospheric disturbance, we introduced a second level of complexity consisting of the application of random, four–directional wind forces that act on the UAV as an additional force shifting its position horizontally or vertically.

Once all these described features were successfully deployed in the environment, we shifted our focus on the implementation of an agent capable of learning/solving it. Initially, for debugging purposes and for gathering a better understanding on the state dynamics, we introduced an user navigable agent (keyboard agent) that served us well to apply the necessary tweaks to the structure of the environment to ensure its correct behaviour.

During the training of the actual agents that we implemented exploiting traditional Reinforcement Learning approaches, we discovered just how crucial the trade–off exploration/exploitation is. We started off by not considering an ϵ–decay and the result was that the agent did not perform any sort of exploration in the environment. Once we introduced it and tuned the corresponding hyper–parameters, we immediately noticed how the learning patterns of the agents improved, especially if Q–learning based.

Chapter 4

Conclusion

In this report we provide details and insights into how we designed the concept of a Reinforcement Learning environment from scratch, implemented it such that the required features are satisfied and deployed an agent attempting to solve it.

When looking at the concept, most of its features were ultimately introduced in the final version of the environment, except for the financial components. As mentioned in Chapter 1, one important aspect of the environment was to train the agent to reduce the usage of the engines as much as possible while still reaching the target. This aspect was indeed incorporated in the reward function of the prototype that was split into a primary and a secondary agent, but due to the issue we encountered while implementing two separate agents (see Section 2.4) we ultimately decided to move forward with a single one. Another important aspect of the environment from a financial point of view, was the idea of training the agent such that it reaches the target swiftly, but due to the same reasons we did not include the reduced engine usage feature, we were also not able to include the optimization of the time feature.

Specifically on the topic of splitting the agents, we proposed in our intermediate report an agent framework consisting of two agents working parallel to achieve the goal of navigating the UAV to a desired target location. The concept was not further pursued in the final version of the environment due to unexpected horizontal shift phenomena of the implemented PID controller, as mentioned in more details in Section 2.4. What we were able to show is that holding a certain state with an PID controller is possible under the assumption that the drone has no velocity along x-direction, meaning that from a physical point of view introducing thrust and drag caused certain discrepancies with the behaviour of the PID. Introducing PID in the conceived environment requires further investigation, especially regarding the stability of the x-position of the drone.

Regarding the training of our agents, we exploited traditional Reinforcement Learning techniques to solve the deigned environment. While a certain degree of learning was performed by the agents, especially the Q–learning based agent, after a number of episodes the agent instead of taking the final leap and truly learn the behaviour of the environment it would stagnate. From a practical point of view, the agent does perform the take–off procedure in safe fashion and it shows tendencies of trying to reduce the distance to the target, nevertheless the task is only solved approximately 15 % of the times and without the addition of any complexities. There could be a large number of reasons as of why the agents are not capable of fully learning the environment; the problem requires further testing and investigation.

In this report, we show how we developed a two–dimensional OpenAI–like environment that successfully accounts for all the Reinforcement Learning features and in terms of physical behaviors it simulates a real–world use–case correctly. We introduced three layers of complexities, namely layer one is not subject of the influence of the wind forces as well as disturbances in the transmission of the action to be performed, layer two is subject to one of the two disturbances and layer three is subject to both, making the environment more challenging and realistic. Regarding future developments, interesting directions would be to first introduce the financial aspects from the original concept, second include PID controlling to ease the task of the

primary agent and last but not least, identify an agent capable of solving the task when exposed to all three layers of complexity.

Appendix A

Peer review

A.1 Flappy Bird

Regarding the Flappy Bird environment, our positive feedback is as follows:

(i) The task that must be solved in the environment is clear and simple.

(ii) The state space is low–dimensional: [boost_{on}, v, x_{rel}, y_{rel}]. This feature is extremely beneficial for a successful application of our Q–learning based agent since the resulting Q–table will not have irrational dimensions.

(iii) The structure of the reward function is easy and simple to interpret.

(iv) The environment is light-weighted environment, which allows its fast training.

When applying our agents to solve the environment, we isolated a few discrepancies in it:

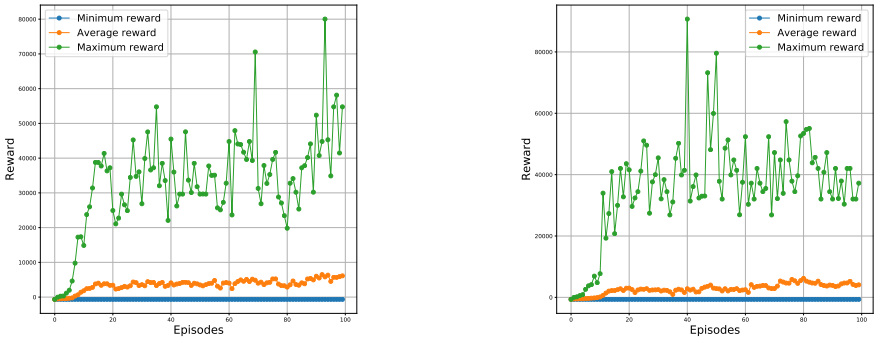
(i) The boundaries of the distance x_{rel}, y_{rel} and the velocity v provided in the observation space do not match the hard–coded limits.

(ii) The boundaries of the discrete observation space are used to initialize the Q table and considering that they do not match the actual output of the environment, one would end up with a state space too large for the kind of problem posed (high degree of exploration would be required to compensate such deficiency).

(iii) The environment as well as the observation/action space lack proper description. There is no explanation regarding the single entries of the observation space, which requires extra time to gather an understanding of the environment before the agents can be applied.

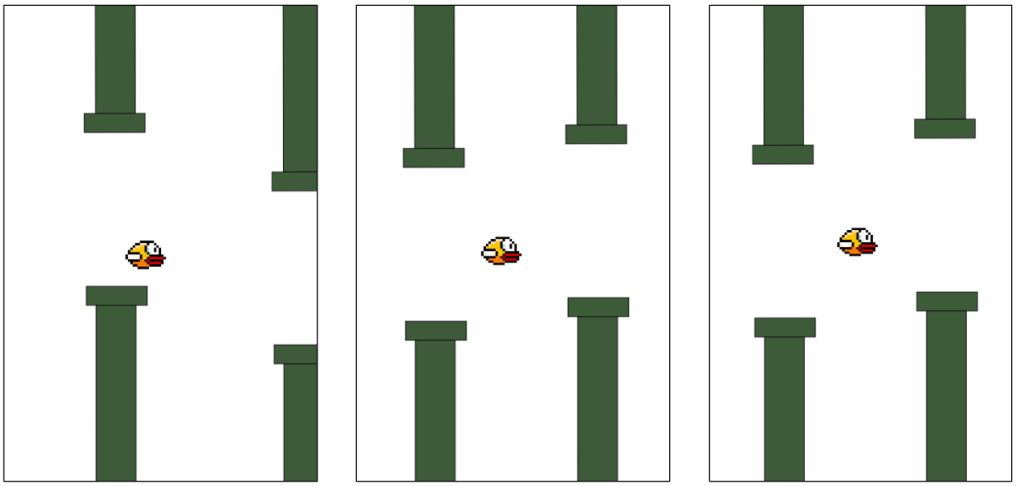
(iv) The introduction of different level of complexities would compliment well the environment.

Despite the described discrepancies in the environment, we managed to apply our agents to it and observe a certain level of learning, especially if Q–learning based. Such rewards can be observed in the reward distribution in Figure FIG. A.1 as well as in the graphical visualization of the learned behaviour in Figure FIG. A.2.



(a) Reward distribution SARSA (b) Reward distribution Q–learning

FIG. A.1: Reward distribution with episode and also its moving progress by using the given discrete observation space as an state space for the Q–learning and SARSA agent algorithm.



(a) (b) (c)

FIG. A.2: Illustration of the learned behaviour. The agent is able to pass several pipes till it fails. The strategy of the agent is to start a boost when a pipe is passed.

A.2 Drone Path Tracking

Regarding the Drone Path Tracking environment, our positive feedback is as follows:

(i) The observation space and the actions are described in great details at the start of the class object, which allowed us to gather a proper understanding of the environment.

(ii) The three–dimensional rendering of the environment excellent.

(iii) The task to be solved has real engineering background as well as substantial use–cases.

When applying our agents to solve the environment, we isolated a few discrepancies in it that ultimately did not allow us to perform any training:

(i) The *reset()* and the *step()* function return high–dimensional arrays that required extensive adjustments of our Q initialization.

(ii) To successfully apply our traditional Reinforcement Learning algorithms without the support of Deep Learning, the continuous state and action space required high–level discretization and adjustments.

(iii) The degree of complexity of the environment as well as the resulting number of possibilities in the continuous action space is substantial.

(iv) The terminal position set at (19, 19, 1) does not match with the boundaries of the overall observation space that are established at (200, 200, 200); if a Q–learning based agent is applied, the size of the resulting Q table would be unnecessarily larger than required, also due to the high–dimensional continuous action space.

(v) The overall drone dynamics appear to be physically incorrect.

A.3 HaxBall

Regarding the HaxBall environment, our positive feedback is as follows:

(i) The observation space and the actions are described in great details at the start of the class object, which allowed us to gather a proper understanding of the environment.

(ii) The two–dimensional rendering of the environment is excellent.

(iii) The structure of the environment can be easily adapted for the application of our Reinforcement Learning algorithms.

As it was the case in Drone Path Tracking, when we applied our agents to solve the environment, we isolated a few discrepancies in it that ultimately prevented us from perform a successful training:

(i) While the continuous state space is per se not problematic, the environment does not provide any information regarding its boundaries; this makes the required discretization an extremely time–consuming task, as the boundaries must be extracted from an extensive simulation of the environment.

(ii) The observation space instance variable return a four–dimensional state space, but *reset()* and *step()* function return instead a ten–dimensional state space.

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] E. E. A. (EEA), E. U. A. S. A. (EASA), and EUROCONTROL, "European aviation environmental report," 2019.

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[4] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[5] G. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

[6] Y. Qin, W. Zhang, J. Shi, and J. Liu, "Improve pid controller through reinforcement learning," in *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, pp. 1–6, 2018.

[7] Kiam Heong Ang, G. Chong, and Yun Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.