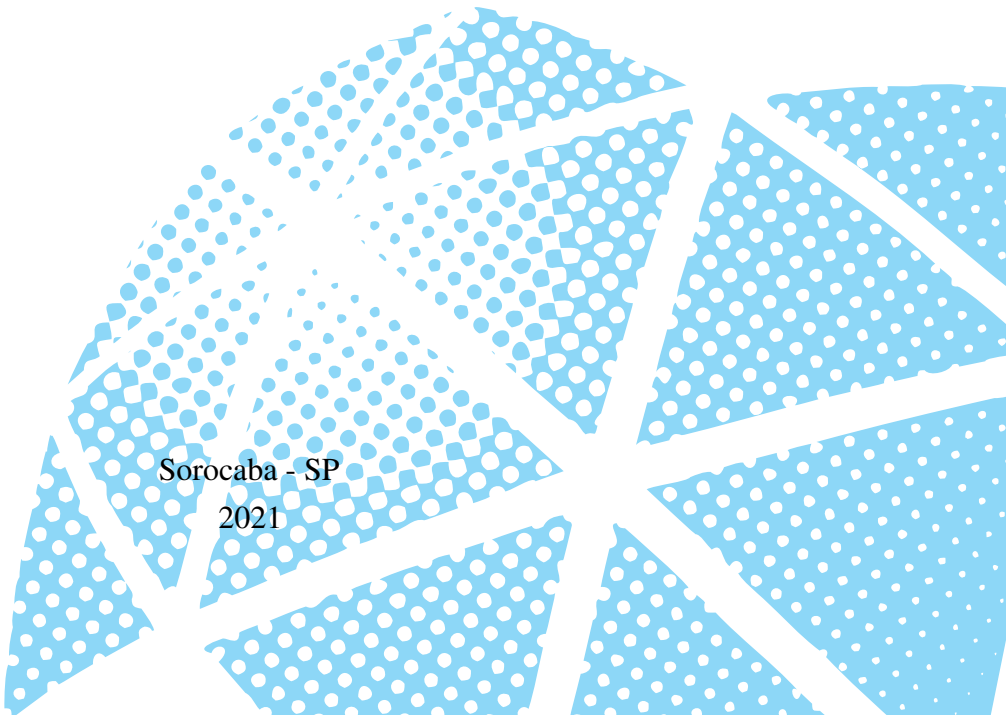
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Ciência e Tecnologia de Sorocaba

ADSON NOGUEIRA ALVES

Control of an unmanned aerial vehicle (UAV) using deep reinforcement learning (DRL) approach



Sorocaba - SP
2021

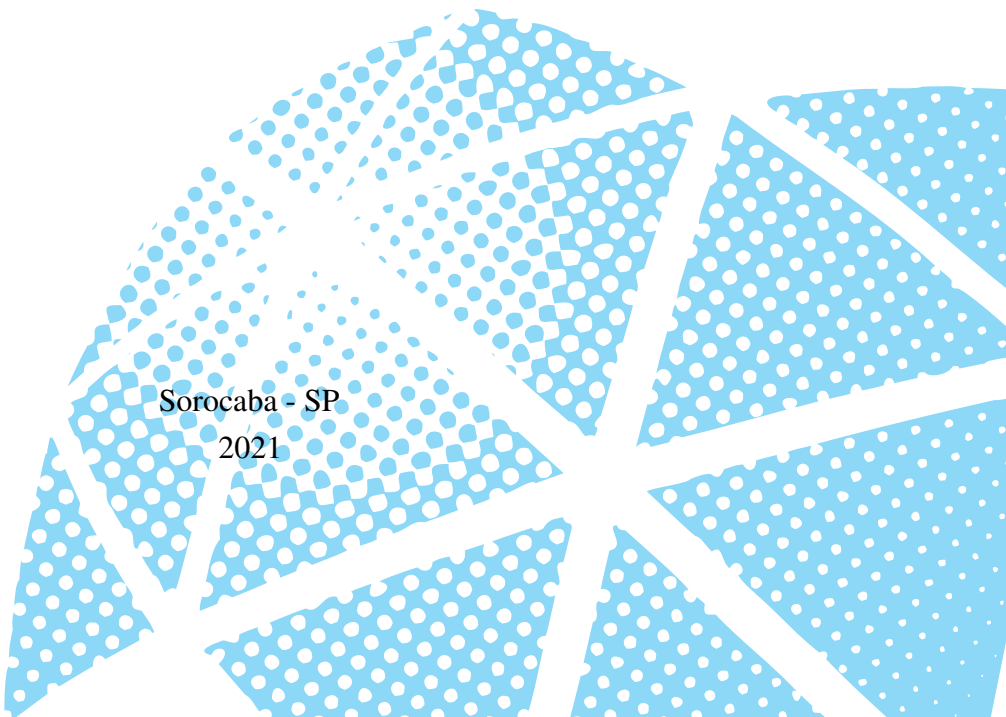
ADSON NOGUEIRA ALVES

Control of an unmanned aerial vehicle (UAV) using deep reinforcement learning (DRL) approach

Text presented to the Graduate Program in Electrical Engineering (PGEE) of the Institute of Science and Technology of Sorocaba as part of the requirements for obtaining the title of Master in Electrical Engineering.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

Supervisor: Prof. Dr. Alexandre da Silva Simões



Sorocaba - SP

2021

A474c

Alves, Adson Nogueira
Control of an unmanned aerial vehicle (UAV) using deep reinforcement learning (DRL) approach / Adson Nogueira Alves. -- Sorocaba, 2021
87 p.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Ciência e Tecnologia, Sorocaba
Orientador: Alexandre da Silva Simões

1. Inteligência artificial. 2. Robot vision. 3. Redes neurais (Computação). 4. Sistemas embarcados (Computadores). 5. Drone aircraft. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Ciência e Tecnologia, Sorocaba. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Sorocaba

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: **Control of an unmanned aerial vehicle (UAV) using deep reinforcement learning (DRL) approach**

AUTOR: ADSON NOGUEIRA ALVES
ORIENTADOR: ALEXANDRE DA SILVA SIMÕES

Aprovado como parte das exigências para obtenção do Título de Mestre em ENGENHARIA ELÉTRICA, área: Automação pela Comissão Examinadora:

Prof. Dr. ALEXANDRE DA SILVA SIMÕES (Participaçao Virtual)
Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia - UNESP - Câmpus de Sorocaba

Prof. Dr. PAULO FERNANDO FERREIRA ROSA (Participaçao Virtual)
Seção de Ensino de Engenharia de Computação / Instituto Militar de Engenharia -IME

Profª. Drª. MARILZA ANTUNES DE LEMOS (Participaçao Virtual)
Departamento de Engenharia de Controle e Automação / Instituto de Ciência e Tecnologia / UNESP / Sorocaba

Sorocaba, 16 de julho de 2021

My advisor and other researchers for sharing knowledge

Acknowledgements

All my family, friends, teachers and employees of the Institute of Science and Technology of Sorocaba, who directly or indirectly contributed to the accomplishment of this work. In particular, I offer my thanks:

- To my parents Adelaide and Nelson for their support;
- To Prof. Dr. Alexandre da Silva Simões and Prof. Dra. Esther Luna Colombini, for all their teaching, encouragement, confidence and guidance;
- To my friends and colleagues at the lab who directly or indirectly helped me.
- The Virtual University of the State of São Paulo (UNIVESP) for the opportunity of teaching professional experience.
- This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

"Only don’t achieve the goal, who dream too far. Only don’t achieve the goal, who intends to take a very long step. Only don’t achieve the goal, who beliaves that things are easy, all things are hard, all things must be fight and when you get something easy, wary."

Senor Abravanel - (Silvio Santos)

Resumo

Veículos aéreos não tripulados (VANT) têm sido alvo de crescente atenção nos últimos anos principalmente devido a sua amplitude de aplicação em atividades complexas e onerosas, como no setor de vigilância, agricultura, entretenimento, entre outros. Todo esse interesse do mercado e acadêmico colocou em evidência novos desafios que a plataforma enfrentará. Entre esses está a complexidade de navegação em ambientes desconhecidos que têm a presença de múltiplos agentes com dinâmica de movimento desconhecida. Novas técnicas de aprendizado têm sido propostas para essas e outras tarefas nos últimos anos. Particularmente, algoritmos livres de modelo baseados no processo de exploração e aprendizado autônomo têm obtido destaque nesse domínio, como é o caso do Aprendizado por Reforço (RL), que busca obter comportamentos apropriados para o robô através de uma abordagem baseada em tentativa e erro e mapeando estados de entrada diretamente para comandos nos atuadores. O presente trabalho busca investigar a navegação de VANTs utilizando um método *off-policy*, o Soft Actor-Critic (SAC), no contexto do Aprendizado Profundo (DL). A abordagem proposta utiliza informações visuais do ambiente e também de multiplos sensores embarcados, bem como o Autoencoder (AE) para reduzir a dimensionalidade das informações visuais coletadas no ambiente. O trabalho foi desenvolvido no ambiente de simulação CoppeliaSim utilizando Pyrep. Nesse cenário, o trabalho investigou a representação dos estados da aeronave e sua navegação em ambientes sem e com obstáculos, fixos e móveis. Os resultados mostram que a politica aprendida foi capaz de realizar o controle de baixo nível do VANT em todos os cenários analisados. As políticas aprendidas demonstraram boa capacidade de generalização, mesmo em ambientes complexos.

Palavras-chave: Inteligência artificial. Aprendizado de máquina. Aprendizado por reforço. Visão computacional. Redes neurais artificiais. Sistemas embarcados. Drones.

Abstract

Unmanned Aerial Vehicles (UAV) have received increasing attention in recent years mainly due to their breadth of application in complex and costly activities, such as surveillance, agriculture, and entertainment. All of this market and academic interest has highlighted new challenges that the platform will confront. Among these challenges is the complexity of navigation in unknown environments where there is the presence of multiple agents with unknown movement dynamics. New learning techniques have been proposed for these and other tasks in recent years. Particularly, model-free algorithms based on the process of exploration and autonomous learning have been highlighted in this domain, like the Reinforcement Learning (RL), that seeks appropriate behavior for the robot through a trial and error approach and mapping input states to commands in actuators directly. The present work aims to investigate the navigation of UAVs using an off-policy method, the Soft Actor-Critic (SAC), in the Deep Learning (DL) context. The proposed approach employs visual information from the environment and multiple embedded sensors and the Autoencoder (AE) method to reduce the dimensionality of the visual data collected in the environment. This work was developed using the CoppeliaSim simulator and Pyrep. In this scenario, we investigated the aircraft state representation and the resulting navigation in environments with or without obstacles, fixed and mobile. The results showed that the learned policy was able to perform the low-level control of the UAV in all analyzed scenarios. The learned policies demonstrated good generalization capabilities, even in complex environments.

Keywords: Artificial intelligence. Machine Learning. Computer vision. Artificial neural networks. Embedded systems. Drones.

List of Figures

List of Tables

List of Abbreviations and Acronyms

AI	Artificial Intelligence
UAV	Unmanned Aerial Vehicle
RF	Radio Frequency
SAR	Search and Rescue
ML	Machine Learning
DL	Deep Learning
DRL	Deep Reinforcement Learning
ANN	Artificial Neural Network
MLP	Multilayer Perceptron
CNN	Convolutional Neural Network
DNN	Deep Neural Network
DBN	Deep Belief Network
RBM	Restricted Boltzmann Machine
CDBN	Convolutional Deep Belief Network
FCN	Fully Convolutional Network
MSE	Mean Square Error
RNN	Recurrent Neural Network
RL	Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
SE	State Estimator
NFQ	Neural-fitted Q
DQN	Deep Q-Network
NAF	Normalized Advantage Function

GSP	Guided Search Policy
TRPO	Trust Region Policy Optimization
GAE	Generalized Advantage Estimation
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
A3C	Asynchronous Advantage Actor-Critic
PID	Proportional-Integral-Derivative
IMC	Internal Model Control
SLC	Successive Loop Closure
RLS	Recursive Least Squares
SVSF	Smooth Variable Structure Filter
KF	Kalman Filter
AFC	Adaptive Filter Controller
SMC	Sliding Mode Control
FL	Feedback Linearization
RGB	Red-Green-Blue
PPO	Proximal Policy Optimization
MTRL	Multi-Task Regression-Based Learning
ESDF	Euclidean Signed Distance Field
SLAM	Simultaneous Localization and Mapping
TLD-KCF	Tracking Learning Detection - Kernelized Correlation Filter
ARC	Aspect Ratio Change
GAK-Means	Genetic Algorithm Based K-Means
FANET	Flying Ad-Hoc Networks
AFRL	Adaptive Federated Reinforcement Learning
CTANS	Centralized Task Allocation Network System

GPS	Global Positioning System
GNSS	Global Navigation Satellite System
PWM	Pulse-Width Modulation
SAC	Soft Actor-Critic

List of Symbols

$w_{i,j}$	Weight associated with the input a_i of the neuron i.
α	Learning rate constant.
a_i	Action *i*
s_i	State *i*
r_i	Reward / Punishment to transition *i*
η	Learning factor (decreased at time)
π	Policy
γ	Discount rate
$\mathbb{H}$	Entropy
ϕ	Roll
θ	Pitch
ψ	Yaw
$\dot{\phi}$	Angular Velocity - Roll
$\dot{\theta}$	Angular Velocity - Pitch
$\dot{\psi}$	Angular Velocity - Yaw
ϵ_t	Distance between the target position and the UAV base at time step t
ξ	Vector difference

Contents

1 Introduction

> *" We share information, we create and pass on knowledge. That's the means by which humans are able to adjust to new situations, and it's what differentiates humans from our earlier ancestors, and our earlier ancestors from primates ".*
> [12]

The research of new technologies capable of improving people's quality of life is inherent to human beings. Our ability to think creatively and to imagine novel solutions needed to survive threats proved to be a major asset [12] to humans. Thus, the human brain's complexity is a great asset to the species. Within an increasingly technological world emerged the natural interest in transferring a certain degree of intelligence to machines. The Turing Machine [13] is a typical example of this interest. In this sense, Artificial Intelligence (AI) emerged as a new field in science and engineering, having more notoriety after World War II, earning that name around 1956 [1]. Among the many possible ways to define AI, Raymond Kurzweil said that AI is: *"The art of creating machines that perform functions that require intelligence when performed by people"* [14]. In this scenario, we can define Machine Learning (ML) as a subgroup of these intelligent systems that can improve with experience [5]. Machine learning techniques are used in various applications, such as medical diagnostics, fraud detection, stock market analysis, speech and writing recognition, strategy games, and robotics [15]. The use of machine learning techniques in robots, being more specific in unmanned aerial vehicles (UAV), is the main interest of this research.

The interest in aerial robots has grown significantly in recent years. This notoriety has been growing due to the UAV application's breadth, both in the research area and in daily activities, such as the delivery of goods, public and private security, pest monitoring and control, maintenance, monitoring, entertainment and others. In general lines, recent researches and development have focused on vehicle design [16] [17] [18], navigation and control [19] [20], safety [21] [22], risk assessment [23], telecommunication networks [24] [25] [26], multi-vehicle coordination or maintenance [27] [28] and cargo transportation [29] [30].

Currently, global distribution networks – such as Walmart – are investing in research and development of package delivery systems [29]. According to the patent itself, the method includes loading a product in an unmanned aerial vehicle (UAV), directing the UAV to the delivery location, and communicating with the consumer through a portable device. The product will only be delivered after feeling that the consumer is already in the receiving position and thus can lower the product, thus avoiding interception by third parties. The company has other patents that complement the structure of this project, such as a delivery tower for UAVs to enable the vehicle to land [31] [32] [33]. Other works and research in the area address some models of technologies

that can be used in this type of application, such as the use of laser-guided UAVs [34]. The system would include a navigation system and a sensor that could detect a laser transmission emitted from the surface of a specified location, detecting the frequency and pulse of the laser transmission to identify who is the destination.

Amazon, another giant in the distribution of electronic products, has also invested in delivery systems that use UAVs. The company recently filed a patent application that involves techniques applied to the delivery of packages after being released in flight by a UAV [30]. The goal is that the package can be launched vertically by a moving UAV. The package would also be monitored during the descent by the UAV itself, using radio frequency (RF), making it possible to change the descent path if necessary. The patent does not detail this adjustment.

Other emerging applications of UAVs involve its use in road networks to assist in emergency care caused by road accidents [35]. One of the main proposals is to use the UAV and an emergency ground vehicle to alert vehicles ahead that the ground emergency vehicle is on the way, thus facilitating the vehicle's access to the accident site. Network security was recently addressed since UAV communication is often based on wireless networks, and messages carry private information about people. Today there is no infallible way to protect UAVs from cyber attacks. Recent works [36] propose an additional encrypted communication channel as a mechanism to prevent external attacks.

The use of UAVs to provide communication – for applications in areas with restricted or no communication – is another research focus today. The organization of UAVs in particular topologies could assist in areas of disaster and also in the regions that are far away from a communication infrastructure [37]. The use of aerial vehicles in urban areas could help overcome interference generated by tall buildings or other devices since the topology of the UAVs can be dynamically arranged, and the network could adapt to guarantee the best signal efficiency.

The market of UAVs is over $127 billion [38] [39]. Civil infrastructure is the most significant area, reaching $45 billion. There are expected approximated 100.000 new jobs involving UAVs activities in next years [40]. Business Intelligence expects sales of UAVs to reach $12 billion in 2021 [41]. Other civil applications of UAVs are [38]: search and rescue (SAR), remote sensing, construction and infrastructure inspection, precision agriculture, delivery of goods, real-time monitoring of road traffic, surveillance applications of UAVs, providing wireless coverage. In general, the key challenges found in this cases could be summarized in: charging challenges, collision avoidance and swarming, networking and security.

Regarding the control techniques of UAVs that can allow these aircraft to perform all these tasks soon, the use of **Machine Learning (ML)** techniques is a growing tendency. Some of the new approaches are the use of **Deep Reinforcement Learning (DRL)** [42] or density-based spatial clustering algorithm [43] in the UAVs optimization. An approach addressed to swarming and avoiding collision is shown in [44] with Deep Deterministic Policy Gradient (DDPG) based approach. Other recent works [45] [46] address networking and security based on ML techniques.

In chapter 3 we will discuss in detail these and other works related to UAV control. Still, we realize that there has been a trend towards using Deep Learning (DL) and Deep Reinforcement Learning (DRL) techniques in recent years, motivating a deeper investigation of both.

1.1 Objectives and contributions

To investigate the control of a UAV using DRL, this work has general and specific objectives described in the following sections.

1.1.1 General objective

Our main goal is to investigate the possibility of learning a model-free navigation policy for UAVs using the Soft Actor-Critic (SAC) algorithm and visual information from the environment and multiple embedded sensors. We use an Autoencoder (AE) method to reduce the dimensionality of the visual data collected in the environment, investigating how the aircraft state representation affects navigation in environments with or without obstacles, fixed and mobile.

1.1.2 Specific objectives

To allow this investigation, this works proposes:

1. To investigate the representation of the states of UAVs in the DRL context, particularly focusing on the investigation of state representations that can simultaneously carry visual and other sensors information;
2. To investigate the aircraft navigability in environments with or without obstacles, fixed or in motion.

This work aims to contribute to the generation of new autonomous navigation techniques for aircraft with applicability in unknown environments.

1.2 Text Organization

This work is structured as follows: in chapter 2 the theoretical foundation of artificial intelligence is presented, focusing on machine learning and reinforcement-based techniques. In chapter 3 a review of approaches adopted to control UAVs is presented. Due to the breadth of applications with different purposes, this review is not restricted to flight control but looks to cover distinct application scenarios. In chapter 4 the proposed material and methods for this work are presented, including software and hardware. Results are presented in chapter 5. Finally, conclusions and future works are presented in 6.

2 Theoretical Background

Artificial Intelligence (AI) is:
" The study of mental faculties through the use of computational models. "
[47]
" The study of the computations that make it possible to perceive, reason, and act. "
[48]
" The study of the design of intelligent agents. "
[49]
" ...concerned with intelligent behavior in artifacts. "
[50]

Some of the well-known definitions of **Artificial Intelligence (AI)** [1] group their approaches into four categories: Thinking Humanly, Acting Humanly, Thinking Rationally and Acting Rationally. In general lines, we can understand **Machine Learning (ML)** as a subgroup of artificial intelligence that improves performance with experience [51]. We can also understand machine learning as a computer program to optimize a performance criterion using sample data or previous experiences [6].

The traditional approach to developing an algorithm is based on a system that receives input data and generates the output data. Still, when the output data does not correspond to the expected, it is necessary to reprogram the algorithm and expect the new program to work. In Machine Learning (ML), the paradigm is shifted to a learning algorithm: given a random batch of input data, the system selects the relevant resources and uses it to train the system. In other words, given new input data, we expect the algorithm to achieve the desired output. It is possible to classify learning according to distinct criteria [1] [52]:

- ***Input/output relation:*** *unsupervised learning*, *supervised learning*, *semi-supervised learning*, *reinforcement-learning*;
- ***Data/model relation:*** *inductive learning* and *deductive learning*;
- ***Nature of the algorithms:*** *evolutionary learning*, *deep learning*, *deep reinforcement learning* and so on.

A briefing about this, we can say that in **unsupervised learning** the learning has no teacher. The goal is to be able to identify relations between the data. The main idea is clustering. In **supervised learning** there is the figure of a teacher, i.e., is assigned the correct label to the training examples. The proposal of **semi-supervised learning** is improve the performance of algorithm

through the use of both *labeled* and *unlabeled* data. Already, in **reinforcement learning**, the *agent* learns from reinforcements from environment, it can be rewards or punishments. We can understand **agent** as anything that can be viewed in the environment [1], will be talked more. The **inductive learning** and **deductive learning** are related to the system obtain or refine knowledge through specific information or data, or simply using logic, respectively. However, it is important to highlight that, in inductive learning, new data can modify the knowledge, whereas deductive knowledge is kept. In **evolutionary learning** the technique is applicable to heuristic problems, that is, applicable to solving problems that would not be easily resolved using a polynomial approach. In **deep learning** (DL), the idea is to learn feature levels of increasing abstraction with minimum human contribution [53]. Finally, the **deep reinforcement learning** (DRL), according to [7], can be defined with the use of deep learning algorithms within RL. The DRL and DL will be covered in greater depth in this work. Towards this path, some structures of algorithms that will be important for understanding DRL will be approached.

2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) remain one of the most popular and effective learning algorithms. The inspiration for the approach comes from the brain and its skills, such as information processing, vision, speech recognition, and learning. Understanding how the brain performs such functions would allow us to develop algorithms capable of performing these tasks on a computer [6]. ANNs can be understood as a collection of individual units – the neurons – connected [1]. The properties of the network are determined by its topology and the properties of the neurons. The most typical neuron in ANNs is the *perceptron*. The mathematical model of this neuron is shown in figure 2.1. The value $w_{i,j}$ is the **weight** associated with the input a_i of the neuron i, and the set of weights W is the free parameter that the learning algorithm must properly tune. Each unit in_j calculates a weighted sum of its inputs, as shown in equation 2.1:

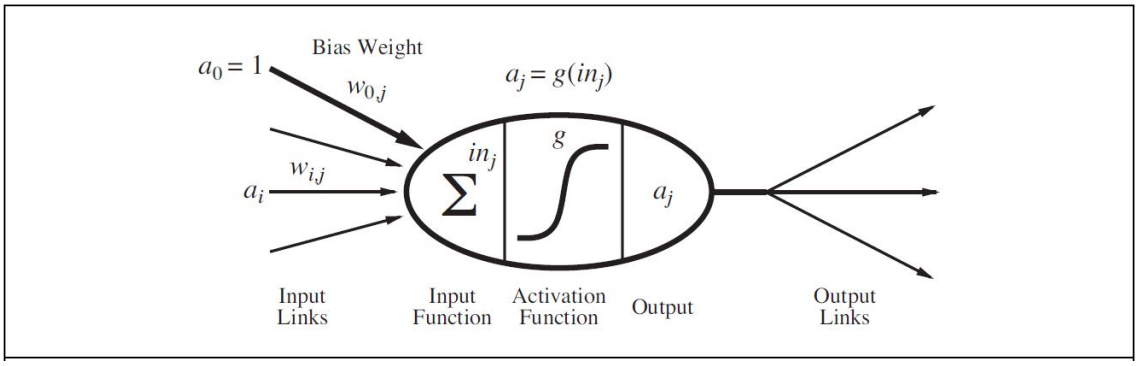


Figure 2.1 – A simple mathematical model for a neuron. The unit's output activation is $a_j = g(\sum_{i=0}^{n} \omega_{i,j} a_i)$, where a_i is the output activation of unit i and $\omega_{i,j}$ is the weight on the link from unit i to this unit. Source: [1].

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i. \tag{2.1}$$

The **activation function** g is applied on this weighted sum to generate the neuron output, as show in equation 2.2:

$$a_j = g(in_j) = g(\sum_{i=0}^{n} w_{i,j} a_i). \tag{2.2}$$

The activation (or transfer) function [54] is responsible for generating the neuron final output value. The *perceptron* typically uses a mathematical function similar to the **threshold function**, and the most usual functions are the **logistic (sigmoid) function** and the **tanh (hyperbolic tangent) function**, both differentiable. The **rectifier transfer function** is also adopted in some cases. These functions are shown in figure 2.2.

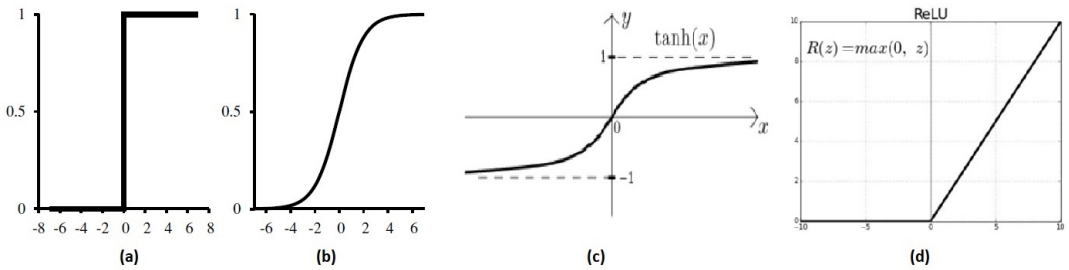


Figure 2.2 – (a) Threshold Function; (b) Sigmoid function; (c) Hyperbolic Tangent function; (d) Rectifier Transfer Function. Source: [1] [2].

The **connection** among the processing units in a network can be made in two distinct ways: the **feedforward network** or the **recurrent network**. In feedforward networks, connections flow in a single direction (from the network input to the network output). In contrast, in recurrent networks, outputs typically feed back into the network inputs. We will employ feedforward networks in this work. These networks are organized in layers, and each unit receives stimuli only from the units that immediately precede it. In a **single-layer neural network** all inputs are connected directly to the outputs. This network is beneficial for processing linearly separable functions like AND and OR but cannot learn a function that is not linearly separable like XOR. We can overcome this limitation by adding a layer between the input and output layers, called the hidden layer. This kind of network, known as **multilayer perceptron (MLP)**, can be a tool for nonlinear regression [6]. If we can calculate the derivatives of the output expressions concerning the weights, it is possible to use the gradient-descent loss minimization method to train the network.

In figure 2.3 we can see a single layer network and a neural network with one hidden layer.

The network is not limited to just one hidden layer. There may be more hidden layers with their respective neurons and weights, thus computing over the values of the previously hidden layer and thus implementing more complex functions. However, with a single hidden layer, large enough [1], it is possible to represent any continuous function of the entries with arbitrary precision, with two hidden layers until discontinuous functions. Some works have shown that when the hidden layer contains many hidden units, it may be wise to add hidden layers, preferring "long and narrow" networks to "short and fat" networks [6].

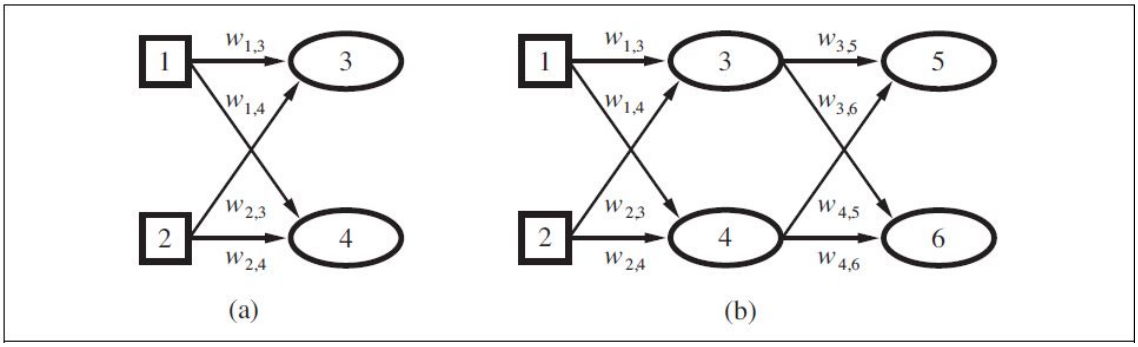


Figure 2.3 – (a) Single layer network; (b) Multilayer network (Multilayer Perceptron - MLP). Source: [1].

The learning in multilayer networks the output vector of a MLP can be express in form [a_i,a_j]. Similarly, a target vector can be [y_i,y_j]. The error found in a_i and a_j depends on all the weights of the input layer, so an update of the weights depends on the errors a_i and a_j. For a loss function L_2, where L_2 is the squared loss function, with a weigth w we have, equation 2.3:

$$\frac{\partial}{\partial w} Loss(w) = \frac{\partial}{\partial w}[y - h_w(x)]^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 \tag{2.3}$$

It is simple to compute the error in the hidden nodes of the network since we only know the expected value in the output layer. Fortunately, we can reflect the error of the output layer for hidden layers. The process is known as **backpropagation**, [55] [56] emerges directly from a derivation of the general error gradient. The backpropagation algorithm can be summarized as [1]:

- Compute the values for the output units, using the observed error.
- Starting with output layer, repeat the following for each layer in the network until the earliest hidden layer is reached:
- Propagate the values back and update the weights

2.2 Deep Neural Networks (DNNs)

Extending this concept, Deep Neural Networks (DNN) are networks with more hidden layers and many neurons in each layer, which is the opposite of a shallow neural network consisting of only one hidden layer. Therefore, DNNs can learn more complicated functions, and their abstraction power increases as the number of hidden layers grows.

Deep learning methods are attractive because the algorithm can discover all that is necessary by itself, assuming that we have a considerable amount of data and enough computation power.

The idea in Deep Learning is to learn features of increasing abstraction levels with a minimal human contribution. This process improves the system dynamics, allowing the automatic discovery of features during training and, therefore, allowing the network to increase its abstraction power and the learning over more general descriptions [53].

Most machine learning methods are described as discriminative, generative, or Hybrid models [57]. According to [58], usually, we can do deep learning as follows.

1. Construct a network consisting of an input layer and a hidden layer with necessary nodes
2. Train the network
3. Add another hidden layer on the top of the previously learned network to generate a new network
4. Retrain the network
5. Repeat adding more layers and after every addition, retrain the network

We present next a summary of mainstream deep machine learning approaches.

2.2.1 Autoencoder

An autoencoder [59] is a neural network with the same number of input and output units, where the number of hidden units is smaller than the number of inputs/outputs. Its training process forces the input data to be equal to the output data, leading the hidden units to represent the input data in a *code* with a reduced number of dimensions. In this way, the first layer acts as an *encoder stage* of the input data, and the output layer acts as a *decoder stage*, reconstructing the original signal from its encoded representation [6].

A MLP with a large number of neurons is usually adopted to implement autoencoders. However, supervised learning is not adopted in this case and is replaced by unsupervised learning since the training process does not require labeled data. In [58] the structure of the learning algorithm can be developed as follows, for each input x:

1. Do a feedforward pass to compute activation functions provided at all the hidden layers and output layers
2. Find the deviation between the calculated values with the inputs using an appropriate error function
3. Backpropagate the error to update weights
4. Repeat the task till satisfactory output

Autoencoder networks are typically adopted in compression and dimensionality reduction tasks and have been particularly useful in image compression. Figure 2.4 shows an autoencoder structure.

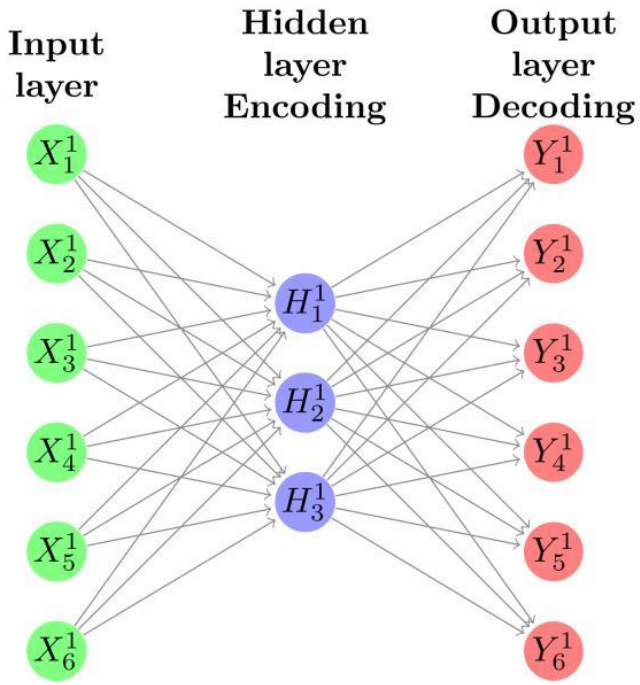


Figure 2.4 – Sparse autoencoder structure. Source: [3].

2.2.2 Deep Convolutional Networks (DCNs)

Convolutional Neural Networks (CNNs), sometimes also called Deep Convolutional Networks (DCNs), were designed for two-dimensional data, such as images and videos, in addition to being the first genuinely successful robust Deep Learning technique [60]. DCNs work by abstracting small pieces of information and combining that deeply into the network. One way to understand it is to imagine that the first layer is responsible for identifying edges of the image, thus forming identification models. The following layers tried to combine this information in simpler forms and, eventually, creating models that vary the object's position, lighting, scale, etc.

Thus the final layers will correspond to the input image with all previous models, and the final prediction is like a weighted sum of all of them [58].

In [4], the author states that pattern recognition by machine involves four primary stages: acquisition, pre-processing, feature extraction, and classification. Feature extraction is usually the most difficult problem to solve, but CNNs offer an adequate alternative, using large sample databases, called training sets. The challenge is to extract features automatically from a portion of the database to allow generalization to other similar images.

The figure 2.5 shows the basic structure of CNN that is fundamental to all of them. One stage of CNN is composed of three volumes: input maps features maps and pooled features maps. The fundamental operation performed at each stage of a CNN is convolution, which justifies its name.

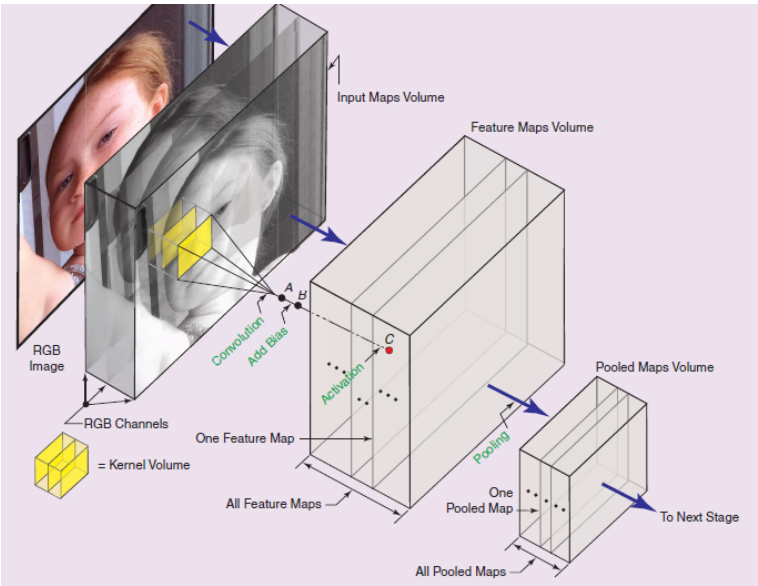


Figure 2.5 – Basic structure of a CNN. Source: [4].

Generally, the volume convolution is performed in CNNs, and there is no change in the volume of the convolution kernel (or filter). It is important to observe in figure 2.5 that the depth of each kernel volume is equal to the depth of the input volume. Thus the volume convolution is simply the sum of the individual 2D convolution. Assuming a kernel volume, the convolution between this kernel and a map of specific features is just the sum of the products of the kernel weights and the map elements that coincide, respectively. The convolutional volume is obtained from the sum of products operation between each respective 2-D kernel (K) since each sum of the product is a scalar. Therefore the K represents the depth of the input volume. The equation 2.4 represent the result of volume convolutional in coordinates (x, y), where w_i and v_i are kernel weights and values of corresponding element, respectively. In figure 2.5 the equation 2.4 represents the result at point A. The point B can be represented by adding a scalar bias, b in

2.4, resulting in a $z_{x,y}$. The point C can be obtained using an *activation function* nonlinear.

$$conv_{x,y} = \sum_i w_i v_i \tag{2.4}$$

The complete feature map, with all *activation values*, is also referred to as an *activation map* has one kernel volume and one bias associated with it. The objective is to learn the weights of each of the kernel volumes and biases through training data. According to [4], a pooled map is simply a feature map of lower resolution. Its method is to replace the values of every neighborhood with the average of the values in the neighborhood. The consequence of this is significant data reduction. Still, the disadvantage is that map size also decreases significantly every time pooling is performed, and when the number of layers is large, it is a problem. Two others pooling methods are the *max pooling* and *L2 pooling*, the first replace the neighborhood value by the maximum value and the second replace with the square root of the sum of their values squared.

Still according [4] the CNNs are structured generally in two ways: a fully convolutional network (FCN) and an image classification. The major application of FCN is image segmentation, i.e., each pixel of an input image is labeled. The FCN can be connected "end to end", allowing the map to decrease first due to convolution and, using an identical network, the reverse process can be done. This allows the output image to be the same size as the input image, but with the pixels labeled and grouped into regions [61]. The image classification is the widest use of CNNs. In this case, the output maps are fed into an FCN to classify it within several predefined classes. The interface between a CNN and an FCN converts 2-D arrays to vectors.

The propagation of a pattern vector towards the output of the neural network is called *Feedforward*. At the same time, the training of a network is done by *feedforward and back-propagation* which is responsible for adjusting the weights and biases throughout the process. Performance can then be measured using an error or cost function. The most commonly used is the mean square error (MSE) between the current and the desired output. The MSE is described by equation 2.5, where $a_j(L)$ is the activation value of j_{th} neuron in the FCN output layer.

$$E = \frac{1}{2}\sum_{j=1}^{n_L}(r_j - a_j(L))^2. \tag{2.5}$$

The training aims to adjust the weights and biases whenever an error classification is found, thus minimizing errors in the output. This is done using gradient descent for both, equations 2.6 and 2.7, α is the *learning rate constant*.

$$w_{ij}(l) = w_{ij}(l) + \alpha\frac{\partial E}{\partial w_{ij}(l)}. \tag{2.6}$$

$$b_i(l) = b_i(l) + \alpha\frac{\partial E}{\partial b_i(l)}. \tag{2.7}$$

2.3 Reinforcement Learning (RL)

In some applications, the system output is obtained after a sequence of actions. In these cases, the important thing is not the immediate result but the policy adopted to achieve the objective through a sequence of correct actions. As it is complex to evaluate the best action in an intermediate state of the system, the machine learning algorithm must learn and evaluate the actions taken, leading to the choice of the best sequence of actions that led to the final objective. Reinforcement learning (RL) can be defined as:

> *" In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives a reward (or penalty) for its actions in trying to solve a problem. After a set of trial-and-error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward. "*
>
> [6]

The *agent* needs to receive a *reward* when it reaches or gets closer to the goal and receive a *punishment* when it deviates from it; hence the term *reinforcement* which can be received during or at the end of the process, will depend on the application, so an optimal policy maximizes the reward received [1].

2.3.1 Observable States

The figure 2.6 illustrates an agent that interacts with the environment by executing a *action* (a_i) that takes to a new *state* (s_i), thus receiving a immediate *reinforcement* (r_i) (reward / punishment) for this transition. Such is the setting of **reinforcement learning**.

Unlike the past methods where we typically had a *teacher*, now learning is done with a *critic* that, unlike the supervised method, is not known the right action, just how well we have been doing in the past. In a simple RL problem where there is only one state and a finite number of possible actions, the value of our action $Q(a)$ is quickly known. If the reward is deterministic, we have $Q(a) = r_a$, so to maximize the value we choose the $max_a Q(a)$. On the other hand, if the reward is stochastic we define $Q_t(a)$ from the probability distribution $p(r|a)$ at time t, the equation 2.8 defines an online update.

$$Q_{t+1}(a) = Q_t(a) + \eta[r_{t+1}(a) - Q_t(a)] \tag{2.8}$$

Where:

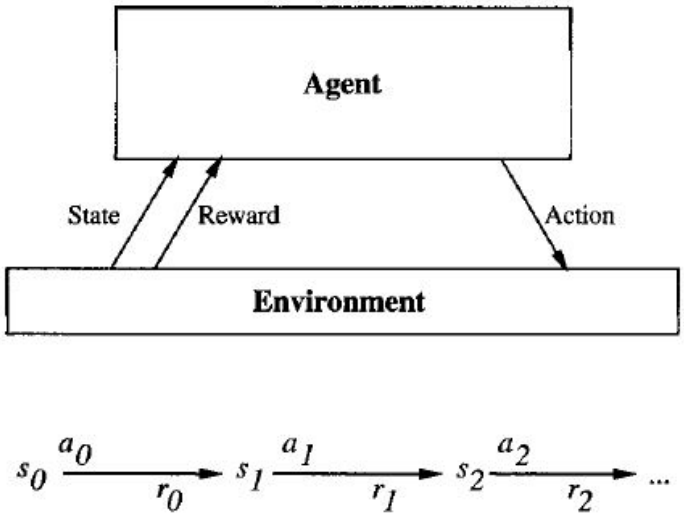


Figure 2.6 – An agent interacting with the environment. Source: [5].

- $Q_{t+1}(a)$ is the expected value of action a at time $(t+1)$
- $Q_t(a)$ is the current prediction
- η is the learning factor (decreased at time)
- $r_{t+1}(a)$ is the reward received after taking action a at time $(t+1)$

The RL problem is modeled using ***Markov Decision Process (MDP)*** where the rewards and the next state are based on the respective probability distribution $p(r_{t+1}|s_t, a_t)$ and $P(s_{t+1}|s_t, a_t)$, depending only on the current state and action. The sequence of actions from initial state to terminal state is an ***episode*** or a ***trial***. The ***policy*** defines the behavior of the agent, that is, the action taken in any state $s_t : a_t = \pi(s_t)$. The value of the policy represents the expected cumulative reward as long as it remains on the policy, $V^\pi(s_t)$, starting at the state s_t.

We can work with models of finite or infinite episodes. For finite models the value of policy π is showed in equation 2.9 and infinite equation 2.10, where T is the next step and $0 \leq \gamma < 1$ is the *discount rate*.

$$V^\pi(s_t) = E[\sum_{i=1}^{T} r_{t+i}] \tag{2.9}$$

$$V^\pi(s_t) = E[\sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}] \tag{2.10}$$

Known as ***Bellman's equation*** [6], equation 2.11, works with the state-action value, $Q(s_t, a_t)$ which denotes how good the performance of a_t in the state s_t, instead than denoting how

good it is for the agent to be in the state s_t, as is the case with $V(s_t)$ seen previously. The policy π is taking the action a_t^* that give us the highest value of $Q^*(s_t, a_t)$. According [7] it's similar to V_π, except that the initial action a_t is provided and π is only followed from the succeeding state onward .

$$Q^*(s_t, a_t) = E[r_{t+1}] + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \tag{2.11}$$

In a ***model-based learning*** all parameters of the environment model are known, and there is no need for exploration once we can solve it through dynamic programming. However, the most practical application of reinforcement learning is when we do not have the model: (***model-free learning***). The ***temporal difference learning*** considers the value of the next state and the reward for updating the current state value. An exploration strategy is based on randomly choosing an action in the number of options, using search $\epsilon - greedy$ with probability ϵ. To continue exploring indefinitely when we have enough exploration, we start exploitation with a high ϵ value and gradually decrease it. Figure 2.7 illustrates a simple deterministic world. Notice that each grid represents a state, the arrows represent possible actions and their reward value, and G represents the goal. In this scenario, equation 2.11 is reduced to equation 2.12. In non-deterministic cases, we use equation 2.11, where the same state and action can lead to different rewards and new states; thus, it is important to keep a running average. This is known as the ***Q-learning*** algorithm.

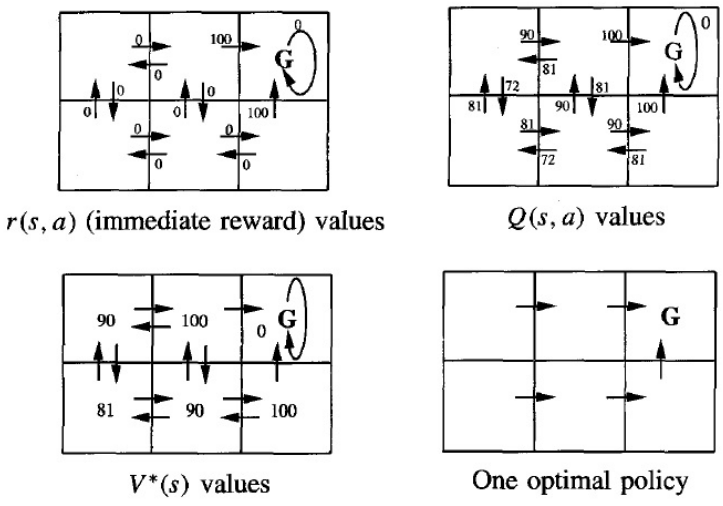


Figure 2.7 – A simple deterministic world. Source: [5].

$$Q(s_t, a_t) = r_{t+1} + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{2.12}$$

On-policy methods estimate the value of the policy used to select the agent's behavior. In ***off-policy methods***, the behavior policy selects actions, whereas another policy, the estimation policy, is evaluated and improved. The Q-learning on-policy version is the ***Sarsa algorithm***.

In some applications it is not possible to store the $Q(s, a)$ or $V(s_t)$ in a lookup table due to a large number of states and actions or situations where the discretization of the data results in an error or still the search space size. In these cases, according [1], it's interesting to consider this as a regression problem, $Q(s, a|\theta)$, with s and a inputs and parameterized by θ to learn the Q values.

2.3.2 Partially Observable States

In some applications, the agent does not know the status exactly, but it can receive indications that lead to predicting the most probable state. This can be done through sensors, cameras, and so on. Despite the similarity with the MDP, the difference is that after acting a_t, the new state s_{t+1} is not known. However for an observation o_{t+1} we arrive at a stochastic function $p(o_{t+1}|s_t, a_t)$ called ***partially observable MDP (POMDP)***. The action is multiplied by the probability of the possible states that are added to the end. However, the state uncertainty can lead to loss of performance that is measured by the cumulative reward. In this case, the use of *Recurrent Neural Networks (RNNs)* can be interesting for maintaining the state and not forgetting past observations.

Actions can take place to get information, thus reducing uncertainties, this is known as ***value of information***. According [6], the agent uses an internal belief state B_t that considers your experiences, this *state estimator* updates B_{t+1}, based on observations, actions and previous belief states. The figure 2.8 illustrates what was said, *state estimator (SE)*, that keep a internal belief state b applying the policy π.

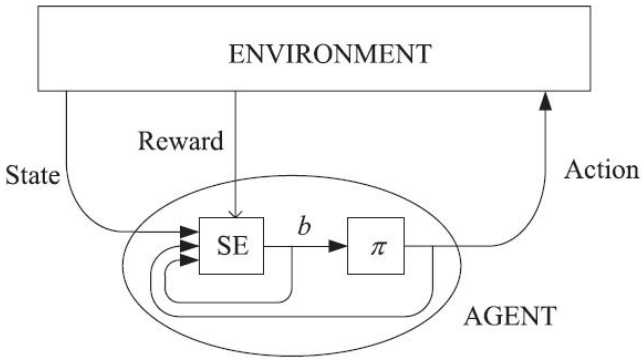


Figure 2.8 – Partially Observable Environment. Source: [6].

The belief state-action pair values is show in equation 2.13.

$$Q(b_t, a_t) = E[r_{t+1}] + \gamma \sum_{b_{t+1}} P(b_{t+1}|b_t, a_t)V(b_{t+1}) \tag{2.13}$$

Instead of ***bootstrapping value functions*** using dynamic programming methods, ***Monte Carlo method*** is a method that estimates the return from the average of several policy implementations, and can be applied in non-Makovian environments. The best of both methods combines *TD* learning and Monte Carlo policy assessment.

Another method shown in [7] is the ***Actor-Critic methods*** that combines the value function and an explicit representation of the policy. Figure 2.9 shows the actor-critic setup. The *actor (policy)* and the *critic (value function)* receives a state from the environment. The actor acts, and the critic, using the reward resulting from the previous interaction, uses the *TD error* calculated to update itself and the actor.

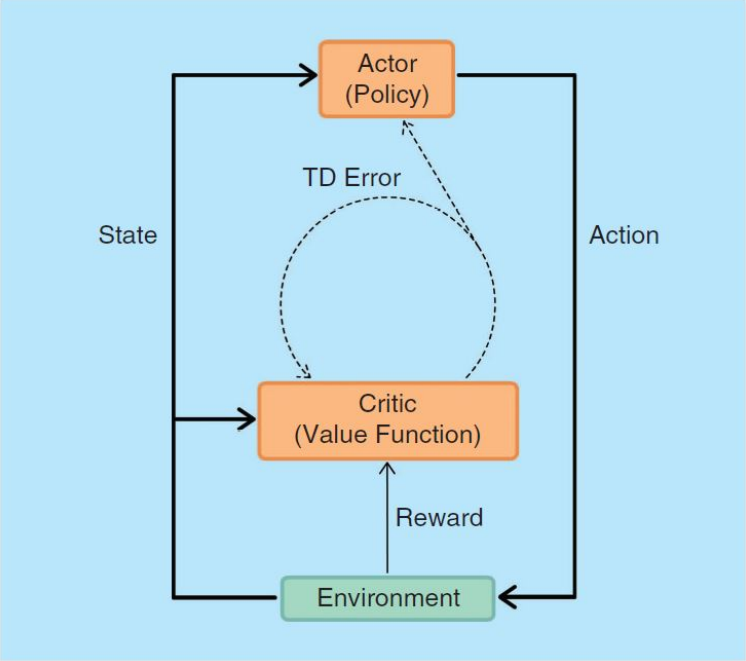


Figure 2.9 – The Actor-Critic setup. Source: [7].

2.4 Deep Reinforcement Learning (DRL)

Despite the practical application of the RL technique, according to [7] it still lacked scalability and was limited to low-dimensional problems due to computational, sample, and memory complexities. However, with DL, these limitations can be overcome, with their use within the RL defining the field of DRL.

2.4.1 Deep Q-network (DQN)

According [7], *DQN* was the first RL algorithm that worked from raw visual inputs in several environments. It emerged from *neural-fitted Q (NFQ)* that combined a deep autoencoder to reduce the dimensionality of the inputs with a separate branch to predict Q-values, as shown in

[62]. To allow for a better choice of actions, $argmax_a Q^\pi(s,a)$, after a single forward pass of the network, allows the network to encode action-independent knowledge in the lower, convolutional layers. With the simple objective of maximizing the reward, DQN learns to extract salient visual characteristics, jointly coding objects, movements, and interactions. The strength of the DQN is in the ability to compact high-dimensional observations and the Q-function using deep neural networks. According to [7] DQN addresses fundamental problem of instability through function approximation in RL using two techniques: ***experience replay*** and ***target networks***.

Experience replay memory reduces the number of interactions with the environment and reducing the variance of learning updates through sampling batches of experience. The transition storage has the form $(s_t, a_t, s_{t+1}, r_{t+1})$ in a cyclic buffer, enabling the RL agent to sample from and train on previously observed data offline. Some works [63] showed that prioritizing samples based on errors *TD* is more effective than uniform sampling for learning. The **Target network** starts with the weights of the network that implements the policy. Still, instead of calculating the TD error based on its estimates of Q values, the policy network uses the fixed destination network. During training, the weights of the target network are updated to match the network policy after a fixed number of steps. One of the main benefits of **DQN** is the function approximator for the Q-function, generating significant improvement in **RL**. So, the Q-learning rule can be updated using a single or double estimator or even using the target network from the DQN algorithm that generates a better result with small updates.

Another way to adjust the DQN architecture is to decompose the Q-function into meaningful functions, that is, to calculate the state-value function V^π and advantage function A^π in separate layers [64]. The ***dueling DQN*** benefits from a single baseline for the state (V^π) and easier-to-learn relative values (A^π). The combination of *dueling DQN* and *experience replay* is one of the state-of-the-art techniques in discrete action settings. Another modification of the DQN that made it possible to work over sets of continuous actions is the ***normalized advantage function (NAF)*** algorithm, being one of several state-of-the-art techniques in continuous control problems [65].

2.4.2 Policy search

Gradient-free or **gradient-based** methods are commonly used as policy search methods. Several successful DRL methods have chosen to use the ***evolutionary algorithms***, according [7], which can be used to train large networks, becoming the first deep neural network to learn an RL task [66]. The interest in evolutionary methods for RL is justified because it can potentially be distributed on larger scales than techniques that depend on gradients.

The **backpropagation** is the basis of DRL, allowing neural networks to learn stochastic policies, computing the loss gradient and weights of the network for a single input-output example. It can help, for example, to decide where to look in an image, which reduces the necessary computational resource. The use of RL to make stochastic decisions over inputs is known as *hard*

attention with many applications outside traditional RL domains.

Searching for a network with multiple parameters can be extremely difficult in addition to suffering from multiple locations. To work around this problem, one way would be to use a **guided search policy (GSP)** which takes advantage of some action sequences from another controller. Thus, through supervised learning and considering the importance of the sample, it is possible to minimize cost and optimize the policy, using a **region of trust** to avoid that the policy update deviates too much from the current one. In this line of work we have the **Trust Region Policy Optimization (TRPO)** [67] applicable for high-dimension inputs. Combined with the **generalized advantage estimation (GAE)** [68] technique it can be very useful in continuous control.

Application with **DRL critical-actor methods** proved effective in real robotic visual navigation tasks through the image pixel [69]. In this context, **deterministic policy gradients (DPGs)** extend the standard policy gradient theorems for stochastic policies to deterministic policies [7]. DPGs integrate only over the state space, requiring fewer samples in problems with large areas of action. Unlike the stochastic policy gradients that integrate over the spaces of state and action, again in that context, the **deep DPG** uses neural networks to operate at high dimensions. Another very popular and recent DRL technique is the **asynchronous advantage actor-critic (A3C)** that combines the advantage of the actor-critic, the asynchronously updated policy and value function networks trained in parallel over several processing threads. A structure to train several DQNs in parallel, obtaining better performance and reduced training time. Another interesting approach is when the agent learns from the demonstration, this is known as **behavioral cloning**.

2.4.3 Soft Actor-Critic (SAC)

The Soft Actor-Critic (SAC) was introduced by[70]. According to the authors, it is an off-policy actor-critic DRL algorithm based on the maximum entropy reinforcement learning framework. The actor aims to maximize expected reward and the entropy, combining off-policy updates with a stable stochastic actor-critic formulation, outperforms prior on-policy and off-policy methods.

The RL standard is that the sum of the reward is maximized, so:

$$\sum_{t} E_{(s_t,a_t)\to\rho_\pi}[r(s_t,a_t)] \tag{2.14}$$

The SAC consider a more general maximum entropy (see e.g [71]), equation 2.15. The α determines the relative importance of the entropy term.

$$J(\pi) = \sum_{t=0}^{T} E_{(s_t,a_t)\to\rho_\pi}[r(s_t,a_t) + \alpha\mathbb{H}(\pi(.|s_t))] \tag{2.15}$$

[70] shows that soft policy iteration converges to the optimal policy within a set of policies that might correspond, for instance, to a set of parameterized densities. And that, large continuous domains require us to derive a practical approximation to soft policy iteration. To do this, they used function approximators for both the Q function and the policy. The soft value function is trained to minimize the squared residual error through more complex calculations that are presented in his work.

To understand the skills acquired through maximum entropy in the reinforcement learning (RL) scenario, it is important to remember that RL employs a stochastic (π) policy to select actions, and thus find the best policy that maximizes the cumulative reward that is collected through an episode of length T, Equation 2.16:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} r_t \right] \tag{2.16}$$

Thus, conventional RL approaches use a unimodal distribution policy centered on the maximum Q-value exploring its neighbor within the probability function. Refining learning policy to the most promising state and ignoring the least likely states. Imagine that in Figure 2.10, the gray curve represents two high-level decisions that the agent must make. The red distribution specifies traditional RL approaches.

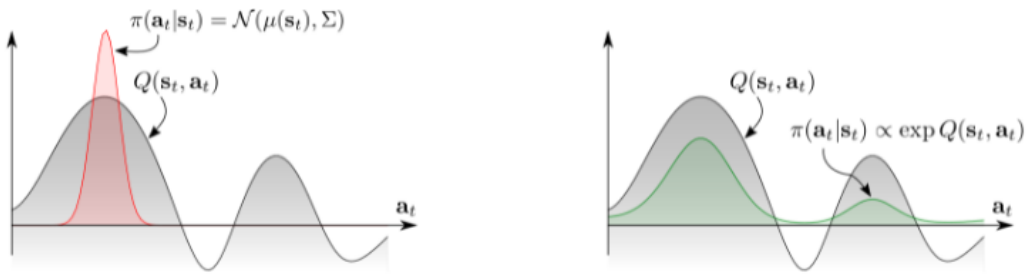


Figure 2.10 – A multimodal Q-function. Extracted from: [8]

Another high-level solution would be to ensure that the agent explores all promising states, prioritizing the most promising state. The formalization of this idea can be given in Equation 2.17, which defines the policy directly in terms of the exponentiated Q-values, represented by the green curve in Figure 2.10.

$$\pi(a|s) \propto exp\, \mathbb{Q}(s, a) \tag{2.17}$$

We can show that the policy defined through the energy form is an optimal solution for the maximum-entropy, Equation 2.18 RL objective, which simply augments the conventional RL objective with the entropy of the policy [72].

$$\pi^*_{MaxEnt} = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t=0}^{T} r_t \ + \mathbb{H}(\pi(.|s_t)) \right] \tag{2.18}$$

An organized description of the algorithm was made by [73], [74] and [75], the Algorithm 2.1 will be adopted in this work.

Algoritmo 2.1: SAC - Soft Actor-Critic

Initialize parameter vector (networks) ψ, $\overline{\psi}\theta$, ϕ ;
for *each epoch* **do**
  for *each environment step* **do**
    $\mathrm{a}_t \sim \pi_\theta(\mathrm{a}_t|s_t)$
    $\mathrm{s}_{t+1} \sim \ \shortmid(s_{t+1}|s_t, a_t)$
    $\mathrm{D} \leftarrow \mathbb{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end
  for *each gradient step* **do**
    $\psi \leftarrow \ \psi - \lambda_V \overline{\nabla}_\psi \mathbb{J}_V(\psi)$
    $\theta_i \leftarrow \ \theta_i - \lambda_Q \overline{\nabla}_{\theta_i} \mathbb{J}_Q(\theta_i) \ for \ i \in \{1, 2\}$
    $\overline{\psi} \leftarrow \mathbb{T}\psi \ + \ (1 - \mathbb{T})\overline{\psi}$
  end
end

3 Related Work

The task of controlling a UAV usually refers to a lot o different challenges (stability, trajectory following, path planning, obstacle avoidance, prediction, etc.) encountered in many different scenarios and for which many other techniques have been applied. In this way, approaches for controlling UAVs can be grouped in many different ways. This section presents a review of the most recent techniques grouped in the following way: i) classical approaches; ii) intelligent approaches.

The **classical approaches** are usually more close to the *control theory* and related techniques. In this context, a usual research focus is the stability control problem. Classical techniques such as PID and Internal Model Control (IMC) [76] [77] are very useful, but they depend on prior knowledge of the system model. Techniques such as Successive Loop Closure (SLC) can be applied together to the PID to adjust the gains [78]. Wen considering the wind on stability problems, the H^2 optimal control theory has been applied [79] achieving satisfactory results. Other techniques explored were the Recursive Least Squares (RLS) and Smooth Variable Structure Filter (SVSF) [80] [81] to estimate UAV control dynamics variables, hardware failure detection variables and to prevent cyber attacks. The result achieved by [80] demonstrated a better convergence of estimation by RLS than in SVSF, although both have proven to be effective. Other works [82] applied the Extended Kalman Filter (EKF) in an autonomous multi-rotor system flying in external and unknown environments predicting the UAV trajectory based on empirical data measured with a certain degree error. The EKF is a nonlinear version of the Kalman Filter (KF), a robust prediction control technique. Other works also apply nonlinear control methods generating a more dynamic control system [83] [19]. Other works [83] focus on the application of Adaptive Filter Controller (AFC) in modeling and controlling the stability of UAVs using the Lyapunov Function to satisfy the stability analysis. Another approach [19] adopts control strategies based on Sliding Mode Control (SMC) – a method that alters the dynamics of a nonlinearnonlinear system that forces the system to slide along a cross-section of the system's normal behavior – and the Feedback Linearization (FL) that transforms a nonlinearnonlinear system into an equivalent linear system. The results showed greater robustness to interferences using FL and a faster adjustment using SMC.

All previous approaches can be classified as belonging to classic control, optimal control, and adaptive control. However, in the last years, techniques related to **intelligent approaches** that increase the level of autonomy of the UAVs have aroused. Some works [20] adopts *degrees of truth* to land the UAV, an approach that is possible using a mathematical model based on Fuzzy Logic, achieving satisfactory results. One of the most important tendencies in last years in the intelligent approaches is the use of techniques related to the **machine learning** (like *artificial neural networks* and *reinforcement learning*), that typically aims to improve their performance in

some task with training.

To achieve autonomous navigation in a closed environment, [84] used the Deep Neural Network (DNN) to filter an RGB image provided by a camera attached to the aircraft to allow its navigation in the environment in a controlled manner. A technique that recently become widely used in machine learning approaches is the **Reinforcement Learning (RL)** [85] [86] [87] [88] [89], in some cases used jointly with other techniques like Recurrent Neural Network (RNN), CNN and Fuzzy Logic. In [90] to improve UAV performance, the authors used the Deep Q-Network (DQN) with Noise Injection, applied and tested in a simulation environment. Other works [91] used the Proximal Policy Optimization (PPO) algorithm and stochastic policy gradient to make a quadrotor to learn a reliable control policy. This work shows the viability of using model-free reinforcement learning to train a low-level control of a quadrotor without supervision [91]. According to some authors, the PPO presents a better sampling efficiency when compared to other algorithms like the Trust-Region Policy Optimization (TRPO) [92], besides being much simpler to implement. In [93] the authors developed, according to them, the first open-source neural network-based flight controller firmware, basically a toolchain to train a neural network in simulation and compile it to run on embedded hardware. Despite the evident contribution, the main objective of the work, according to the author, is to improve the altitude control of the UAV traditionally done by a PID controller.

New approaches have aroused in applications like combat and reconnaissance missions. Some works [94] adopted a strategy based on Deep Learning (DL) and Multi-Task Regression-Based Learning (MTRL) for navigation and exploration of forests, regardless of the presence of trails and GPS. The technique consists of two subnets with a convolutional layer each. Some works [95] focused on improving UAV's decision autonomy on battlefields. They applied the Deep Belief Network (DBN) with Q-learning and a decision-making model based on Genetic Algorithms (GA), achieving satisfactory results. Already in the combat context, some works looked to identify who is controlling an opponent aircraft using surveillance images and a CNN architecture to learn human interactions with the relevant object (possible controller) in the scene [96]. Other works focus on the objective of learning reactive maneuvers in one-one aerial combat between UAVs based on the Asynchronous Actor-Critic Agents (A3C) algorithm and RL [21].

When navigating in unknown environments, an autonomous aircraft must have the ability to detect obstacles, thus avoiding a collision. Several methods became available in the literature in recent years. Some works [97] adopted an approach of Deep Deterministic Policy Gradient (DDPG), with continuous action-space, able to train the UAV to navigate through or over obstacles to achieve a target. The DDPG was designed as an extension of deep Q-network (DQN), combining the actor-critic approach with insights from DQN [98]. The reward function was designed to guide the UAV through the best course while penalizing any crash. In [99], the authors applied the free gradient-based planning framework called Euclidean Signed Distance Field (ESDF). It significantly reduces the computational cost since the collision term in the penalty function is

formulated by comparing the collision trajectory with the collision-free guided path, leading to a robust and high-performance algorithm.

In some works [100] [101], looking to allow a UAV to perform an autonomous operation in an internal environment, the Simultaneous Localization and Mapping (SLAM) technique was used through a grid map by Monte Carlo to estimate the 2D position of the vehicle and the map of the environment while moving. The Kalman Filter is used to track the vertical altitude and velocity. In [102] the Kalman Filter was also used, but now to estimate motion and speed in real-time. The proposal is that the UAV can navigate in an external foliage environment without using GNSS, using only a 2D laser range finder. According to the authors, the experiment demonstrated successful autonomous navigation in both indoor and outdoor environments. In [103] the Reinforcement Learning approach is applied now to avoid collisions and investigate the optimal trajectory for the UAV based on the Traveling Salesman problem. In [104] authors adopted a Deep Reinforcement Learning approach using an algorithm derived from POMDP based on the Actor-Critic architecture to allow autonomous navigation in complex environments.

When considering the best trajectory, some approaches [105] uses Q-learning to address the problem, and others [106] use a Dijkstra algorithm together to image processing technique and greedy breadth-first search technique, both achieving good results for outdoor environments. Still considering UAV applications for external environments, some authors focus on target search in complex scenarios based on Optical Flow-Based Method that uses the concept of apparent motion of objects caused by relative motion between an observer, and a scene [22]. This approach proved capable of estimating a rotorcraft 3D position and velocity signals compared to a reference. To enable a UAV to act in a complex disaster scenario, some authors [107] adopted a Deep Reinforcement Learning-based technique inspired by the good results of this technique when applying in an ancient game puzzle Nokia snake.

Other applications such as tracking of moving targets [108] use the vision-based SLAM method, already mentioned in other applications in this work. The author's goal is to use tracking in both indoor and outdoor environments. Another interesting technique is the Tracking Learning Detection - Kernelized Correlation Filter (TLD-KCF) in which a conditional scale adaptive algorithm is adopted [109]. Other Reinforcement Learning approaches [110] were considered together with computer vision techniques to improve the accuracy in UAV tracking considering Aspect Ratio Change (ARC). Results showed to be capable of significantly improve the tracking performance at a low computation cost.

Another important research focus is the joint and collaborative use of these aircraft. Among the possible applications, we can cite wireless internet connectivity, data transfer, and information sharing among UAVs. In most of the works, Reinforcement Learning techniques [111] [112], Deep Reinforcement Learning [26] [113], Deep Deterministic Policy Gradient [114] [24] [115] [28] and Deep Q-Network [116] [25] [117] are the most applied. Other techniques such as Genetic Algorithm Based K-Means (GAK-means) with Q-Learning were used [118] to

allow a dynamic movement of multiple UAVs. The results showed fast convergence with a low number of iterations and better results than other algorithms such as K-means and Iterative-GAK.

Looking to establish mutual attention between an outdoor UAV and a human, that is, a dynamic of mutual interaction between both, some works [119] adopted the Kalman Filter and computer vision techniques. Some authors [120] applied a DNN called TrailNet to maintain the trail center using label smoothing and reward entropy for autonomous navigation on a forest trail alerting users about environmental awareness. The UAV achieved stable and robust navigation, validating the technique.

In wireless networks, the UAV is typically vulnerable to interference that can affect its performance and security. In [121] the authors addressed this problem using the Adaptive Federated Reinforcement Learning (AFRL) - based technique, which proved to be 40% better than other methods used.

Summarizing this literature review, table 3.1 presents the applications in UAVs and the evolution of the adopted control techniques. This analysis shows a clear trend towards using techniques related to the DL and DRL in the last years, stimulating deeper investigation about these techniques.

Table 3.1 – Applications in UAVs and the evolution of the adopted control techniques.

Evolution of the use of techniques for each application				
Technique	Dynamics and Stability Control	Better trajectory and avoid collision	Target location / tracking / recognition	Information Sharing and Connectivity
PID	[76]			
Dijkstra		[106]		
ROSGPS+ CTANS				[27]
TLD+KCF			[109]	
ESDF		[99]		
SVSF+RLS	[80] [81]			
Fuzzy Logic	[20]			
KF		[102]		
EKF	[82]	[101]		
AFC	[83]			
H_2 optimal control	[79]			
SMC	[19]			
EA/GA				[118]
SLAM		[100]	[108]	
FQL	[87]			

RL	[85] [86] [88] [89]	[103]		[111] [112]
Q-Learning	[78]	[105]		[122]
RL+VC			[110]	
CNN	[93]		[96]	
DBN			[95]	
DNN	[84]			
DQN				[117] [25] [116]
DRL	[90]	[104]	[94] [107]	[113] [26]
DDPG				[28] [24] [114] [115]
PPO	[91]			
SAC	[123]			
A3C			[21]	

4 Materials and Methods

This chapter presents the approach proposed to achieve our goals, detailing the UAV dynamics, simulation environment, hardware, agent parameters, models, networks, and algorithm. The experiments proposed are also described.

4.1 Proposed Approach: overview

In this work, we propose to investigate DRL-based algorithms – particularly the SAC algorithm – to train a low-level controller for a quadrotor using a set of **visual and non-visual sensors**. In other words, we propose to investigate the use of visual information together with the multiple sensors embedded in the aircraft to create the state space for the DRL algorithm. A key question in this approach is: *how can we model the system states* (S) to allow accurate control of the UAV?. We will address this question in the section 4.6.

The diagram of the SAC algorithm and the Autoencoder (AE) Network is shown in Figure 4.1, the structure proposed by [75] was used, adding the Autoencoder. The current policy is used to interact with the environment at each epoch of our training, storing inside a replay buffer in the format (s_t, s_{t+1}, r_t, a_t), which is used to estimate a value to the state, a $Q - value$ to transition from $s_t \rightarrow a_t \rightarrow s_{t+1}$, using $Q - value$ to weight our optimizing policy for actions that increase the value of Q. The Autoencoder is used to reduce the dimensionality of 4 images, from 64x64 pixels to 4x4 pixels.

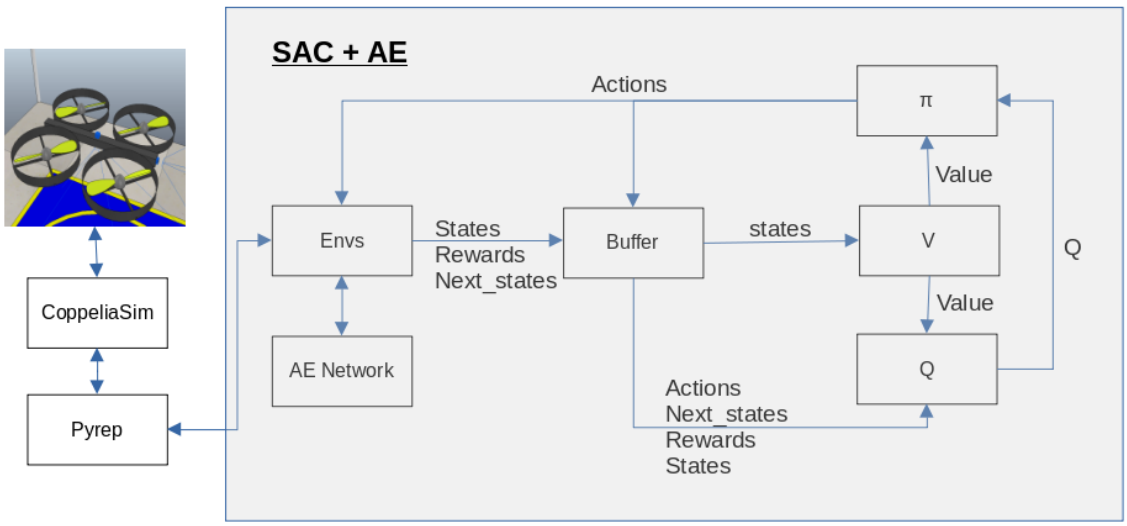


Figure 4.1 – Diagram of the proposed framework using SAC and the Autoencoder.

4.2 Proposed Framework

The work was developed in the Coppelia Simulator [9] and Pyrep framework [124], thus it was possible to increase the simulation performance by speeding up the process by about 20x, compared to the remote API provided by the Coppelia Simulator. The default quadcopter model available in the Coppelia Simulator was used, the UAV mass and the moment of inertia were adjusted to the same used by [75]. using 0.10 kg and $[5.217x10^{-3}, 6.959x10^{-3}, 1.126x10^{-2}]$ kg.m², respectively.

4.3 Coppelia Simulator and Pyrep

The Coppelia Simulator has a wide variety of models in its libraries, mesh manipulation at runtime, and different physical engine options for the user [9], such as:

- Support for platforms: Linux, Windows, and macOS;
- Physics engine used for calculations: Bullet, ODE, Vortex and Newton;
- Outputs: Videos, graphics and text files;
- Library: Wide variety of robots (mobile and fixed), sensors, and actuators;
- Operation with mesh: Allows mesh manipulation at runtime. Imports meshes as element groups, providing flexibility in handling the imported model's materials, appearances, and textures;
- Programming: offers six different approaches.

Coppelia Simulator, in general lines, is a simulation environment that allows testing prototypes and algorithms without involving the constructive costs of a real robot. With its integrated development environment (IDE), it is possible to create scenes to control systems (robots, mats, cameras, sensors, and others) through several scripts in the same scene or through external interfaces. According to Coppelia Robotics, the simulator is an integrated development framework designed for a distributed control architecture. Within a scene, Figure 4.8, it is possible to assign to each object/model, independently, a control encoding in the form of an embedded script, plugin, ROS or BlueZeroNode, remote API, or through a customized solution such as the Pyrep [124]. The Figure 4.2 illustrate these communication modes.

We used the PyRep that is a toolkit for robot learning research, built on top of Coppelia Simulator, plugin sped the process up approximately 20x, compared the other communication modes such as the Remote API, seen in [75].

As explained previously, the default quadcopter model of Simulator was used, Figure 4.3, and all parameters such as mass, the moment of inertia, the velocity-thrust function obtained from

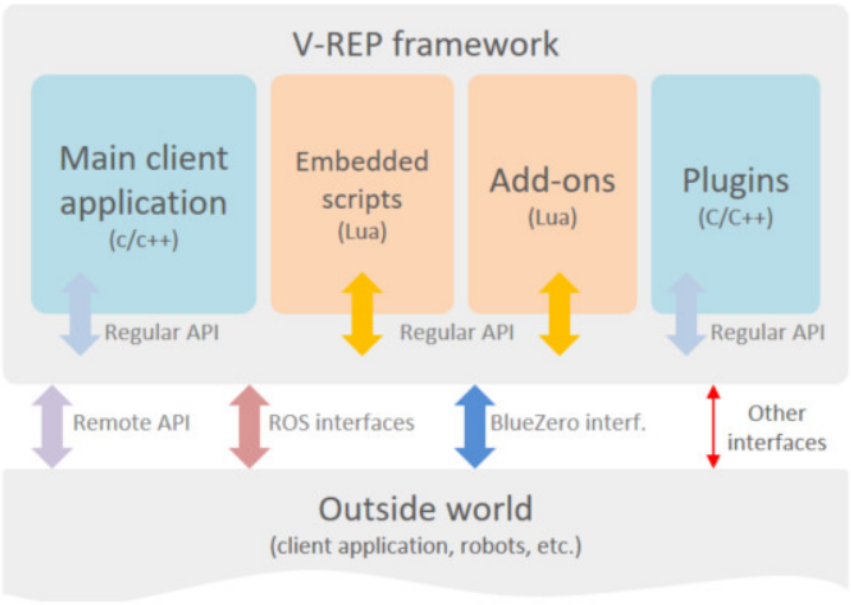


Figure 4.2 – Interfaces to Coppelia Simulator [9].

experiments described in [125], and applied in [91] and [75] will be maintained. The function of propeller thrust force $T_r(pwm)$ is described by equation 4.1.

$$T_r(pwm) = 1.5618 * 10^{-4} * pwm^2 + 1.0395 * 10^{-2} * pwm + 0.13894 \tag{4.1}$$

4.4 Hardware

The experiments were performed on 2 (two) machines and their specifications are:

Machine 1:

- CPU: Intel®- Core^{TM} i7-7700U CPU - 3.60GHz
- RAM: 16GiB
- GPU: NVIDIA - GeForce^{TM} GTX 1080 (8gb)

Machine 2:

- CPU: Intel®- Core^{TM} i7-4510U CPU - 2.00GHz
- RAM: 8GiB
- GPU: Intel - Haswell-ULT Integrated Graphics Controller

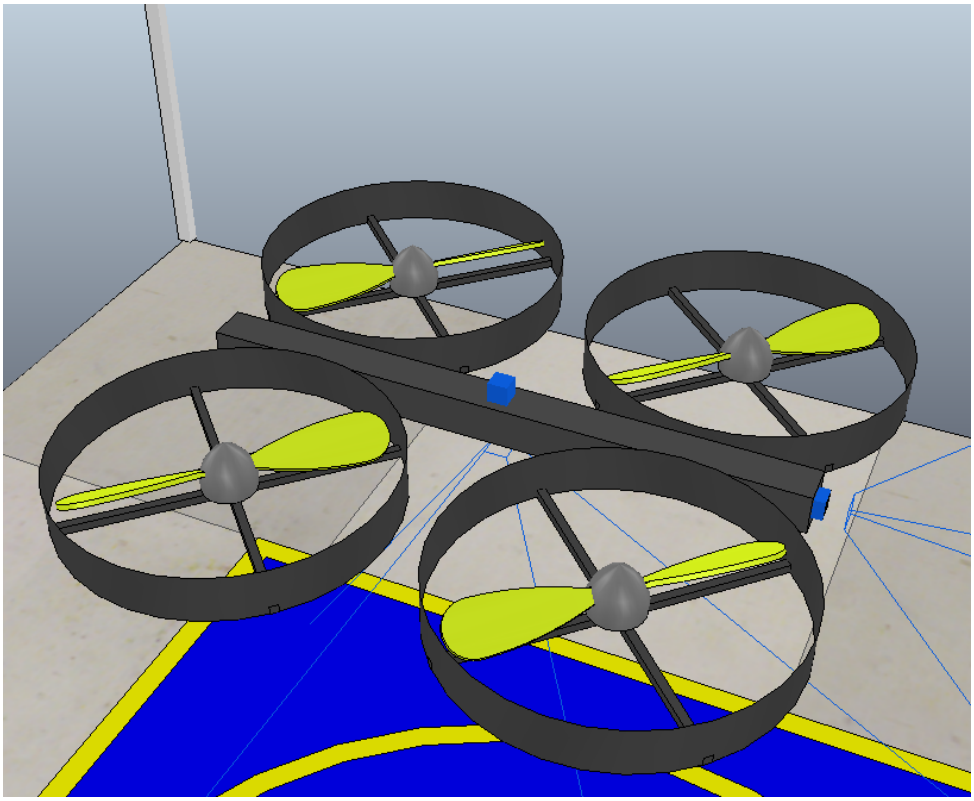


Figure 4.3 – Coppelia Simulator Default UAV - AR Parrot [10].

The library open source chosen of deep reinforcement learning was Pytorch [126], based in Torch library frequently used in vision computation.

4.5 Drone Dynamics

Since we will use a model-free DRL algorithm, the description of all drone dynamics is out of the scope of this work. More details are available at [11], [127], [128]. We are interested in the position, velocity, and angle of the aircraft used in our model variables.

Consider the UAV body frame (B) at the center of the coordinate axis $[X_B, Y_B, Z_B]$, with a weight vector given by mg in the opposite gravity direction, the torque performed in the UAV propellers represented by $[T_1, T_2, T_3, T_4]$ and the angular movement velocity of the propellers, which can be given by vector $[R_1, R_2, R_3, R_4]$. This model structure can be seen in Figure 4.4.

Note that propellers 2 and 3 are on the right side of the X-axis while propellers 1 and 4 are on the left side. We emphasize this because it is important to make sure that the propellers on the same side spin in opposite directions. The propellers diagonally opposites spin in the same direction, i.e., 1 and 3 in one direction, while the 2 and 4 spins in the opposite direction. The algorithm can learn this.

For our approach, we consider that the global position of our UAV in the environment is

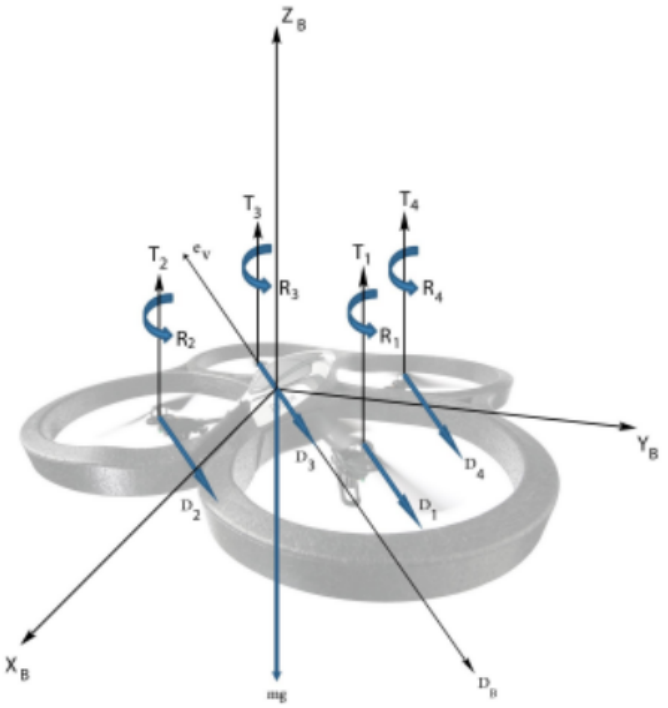


Figure 4.4 – Structure and dynamics of the quadcopter body - Font:[11].

given by $[x, y, z]$, so the linear velocity will be given by $[\dot{x}, \dot{y}, \dot{z}]$. Other important parameters are the Euler angles of the aircraft axes ϕ, θ and ψ, in axes x, y and z, respectively, which are also referred to as roll, pitch and yaw $[\phi, \theta, \psi]$. Consequently our angular velocities are given by $[\dot{\phi}, \dot{\theta}, \dot{\psi}]$. The Rotation Matrix is another important element responsible for convert coordinates from the body frame to the world frame, as can be seen in Equation 4.2.

All computation and logic used are performed within the algorithm developed by us.

4.6 Agents/Models/Networks

4.6.1 Drone Agent

It has been defined that the time horizon of UAV remains until it suffers a reset event, such as collision, go out the global limit, distance from target greater than 19.5 meters or epochs greater than 250 time steps. The standard routine adopted was:

- Reset mode that applies a new initial state or a previous state and can restart the simulation;
- A $shutdown$ method, that stop the episode, when necessary;
- The $global\_limit$ which is responsible for returning if the UAV is within the global limit;

- The *step* method, that is responsible for obtaining and applying new actions on propellers, requesting environment observation states, verifying if the uav reached the objective, weighing the chosen path and receiving the value of the reward function, thus returning these values to the network.

4.6.2 Scenarios

The proposed scenes were built to explore the autonomy of the UAV in different environments. For this, it is important to observe the stability of the aircraft and measure whether it can maintain its stable flight along the trajectory until it reaches the target base.

All scenes have 7 (seven) landing/takeoff bases, [B1, B2, B3, B4, B5, B6, B7], and 4 (four) vertical rods in the corners that sets the limits of the test platform, [corner1, corner2, corner3, corner4]. We will add pipelines and some people to the scene to create scenarios with fixed and mobile obstacles.

1. **Empty environment - SC0.** The first scene is the same one used by [75], the reference used is the green target, a dummy object that serves as a geographic point in the environment and the target position for the aircraft. The worked scene can be seen in the Figure 4.5;

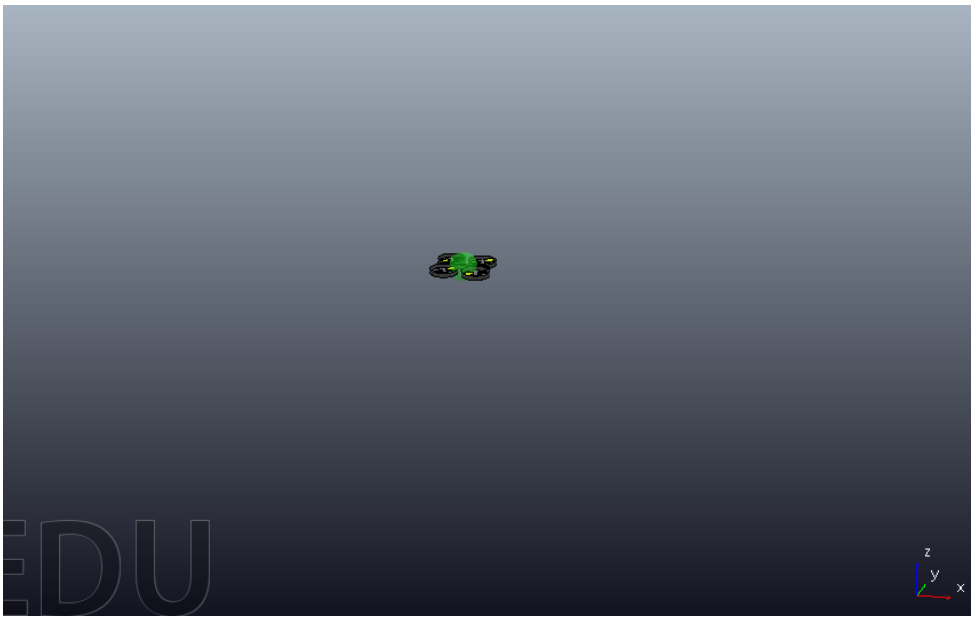


Figure 4.5 – CoppeliaSim Robotics Simulator - Scene empty.

2. **Free environment - SC1.** The second scene intends to investigate the robustness of the flight in a horizontal free displacement. The main behaviors observed were flight stability, accuracy, chosen trajectory, and whether it achieved the objective. The worked scene can be seen in the Figure 4.6;

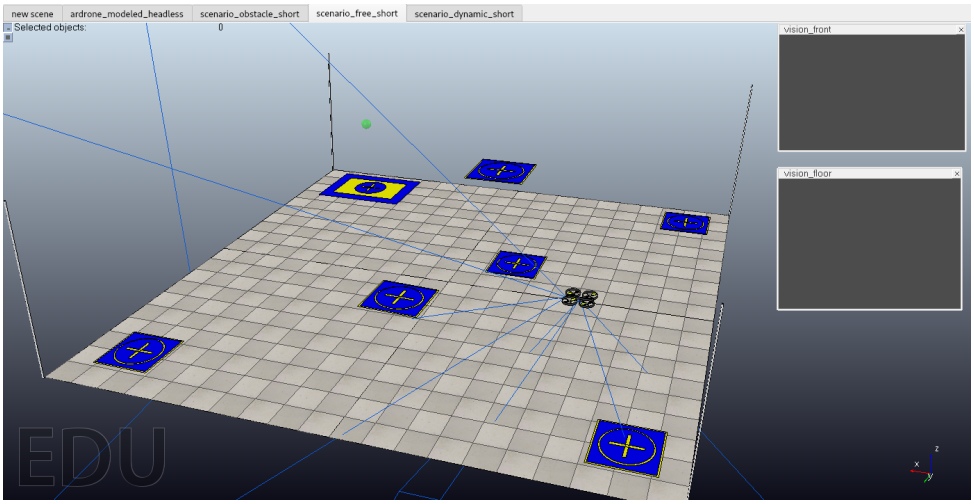


Figure 4.6 – CoppeliaSim Robotics Simulator - Scene free.

3. **Environment with fixed obstacles - SC2.** We will position obstacles (like coastal and land bases, pipes, and so on) in the aircraft's path. With this, we aim to verify the decision autonomy to avoid collisions and maintain an efficient route. The worked scene can be seen in the Figure 4.7;

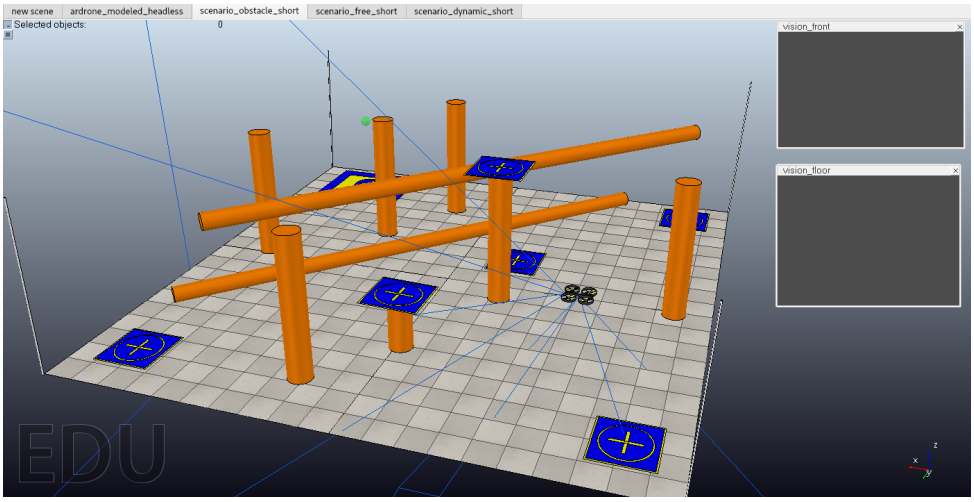


Figure 4.7 – CoppeliaSim Robotics Simulator - Scene with fixed obstacles.

4. **Environment with mobile obstacles - SC3.** This is the hardest challenge for the aircraft. The objective of the UAV is the same as in previous scenarios (to reach a particular destination). Still, obstacles that keep moving – in this case, some people – will be inserted in the trajectory. The proposal is to evaluate the autonomy of the controller under dynamic conditions. The worked scene can be seen in Figure 4.8.

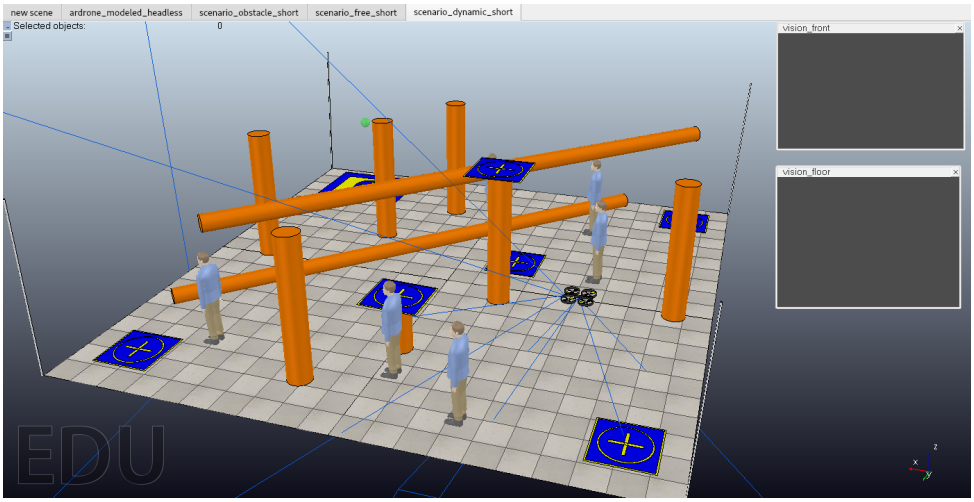


Figure 4.8 – CoppeliaSim Robotics Simulator - Scene with dynamic obstacles.

4.6.3 Representation of states

The representation of states was structured according to the Table 4.1, the 22 states defined in [75] and [91] were maintained and 32 more were added, highlighted by the column accumulated in Table 4.1.

The $UAV\_Position\_Target$ refer the global position of the UAV base relative to target position, represented by the coordinates [*x, y, z*]. Therefore, the *UAV_Linear_Velocity* is defined as [$\dot{x}$,$\dot{y}$,$\dot{z}$]. *UAV_Rotation_Matrix* is responsible for convert coordinates from body frame to world frame and vice versa, it is a scalar product between individual axis rotation matrices, it can be seen in Equation 4.2. We can define that our orientation in the world frame as [ϕ, θ, ψ] (roll, pitch, yaw), that represent the Euler angles of the body axes, thus the *UAV_Angular_Velocity* can be [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$].

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_3 = R_x(\phi)R_y(\theta)R_z(\psi). \tag{4.2}$$

The *UAV_Propellers_Action* represent the actions chosen to stabilize and move the UAV. Distance sensors were added to the aircraft, with one on top of the UAV, one below the UAV, and eight other sensors monitoring around the device structure, distributed equidistantly from each other, thus monitoring a wider area. The sensors were configured to capture any body or object from a distance of three meters with a volume of type randomized ray, where 500 rays will scan a cone-shaped volume at random. For measuring these sensors, we added the *UAV_Ultrasonic_Sensors* in the states. Other important states are: *UAV_Global_Limit* which verifies whether the UAV remains within the pre-defined region for the flight, limited by the corner objects of the scene; *UAV_Travelled_Path* measuring the path taken by the UAV before reaching the target position, suffering a collision, leaving the pre-defined limit or reach 250 time steps. The UAV is also equipped with two monocular cameras in front and below it. The cameras are responsible for capturing images during each instant of time, which has a dimension of 64 x 64 pixels. We propose to use these images to assist the aircraft navigation and to identify obstacles. However, to solve the high dimensionality in the states, we use an autoencoder. The size of each image after the encoder is 2 x 2 pixels. To enable the UAV to recognize the displacement within the environment, we used two images per camera that refer to its last and current frames. Therefore, states *UAV_Last_Floor_Image*, *UAV_Last_Front_Image*, *UAV_Currently_Floor_Image* and *UAV_Currently_Front_Image* for the captured images were added. How we are using an autoencoder, it is important to observe the accuracy of the loss rate in these images, so *UAV_Autoencoder_Loss_Rate* was also considered a state to be observed. Finally, we also consider UAV position relative to the environment as an important state to observe, so the $UAV\_Position\_Env$ has been added.

In general, these were the states used.

Table 4.1 – Representation of states.

Observation States			
Item	States	Number of Elements	Accumulated
1	UAV_Position_X_Y_Z	3	3
2	UAV_Rotation_Matrix	9	12
3	UAV_Angular_Velocity	3	15
4	UAV_Linear_Velocity	3	18
5	UAV_Propellers_Action	4	22
6	UAV_Ultrasonic_Sensors	10	32
7	UAV_Global_Limit	1	33
8	UAV_Travelled_Path	1	34

9	UAV_Last_Floor_Image	4	38
10	UAV_Last_Front_Image	4	42
11	UAV_Currently_Floor_Image	4	46
12	UAV_Currently_Front_Image	4	50
13	UAV_Autoencoder_Loss_Rate	1	51
14	UAV_Position_Env	3	54

4.6.4 Reward function

The reward function is an important parameter in the performance of the learned policy. However, it is not an elementary definition since all UAV elements' abstraction, and their behavior in the environment can be complex. Several attempts were made, considering approaches as:

- Divide the reinforcement into groups, related the proximity of the UAV and the target position;
- Strong punishments for *collision* and *go out global limit*;
- To punish high speed of roll, pitch and yaw [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$];
- To punish long paths to the target position;
- To reward the UAV flight height.

After applying these approaches, unsuccessfully, the best result was still the one used by [91], defined by Equation 4.3, so this approach will be maintained. We take into account stability, robustness and precision.

Thus, the reward function used in this work is defined by Equation 4.3.

$$r_t(s) = r_{alive} - 1.0||\epsilon_t(s)|| - 0.05||\dot{\phi}|| - 0.05||\dot{\theta}|| - 0.1||\dot{\psi}|| \tag{4.3}$$

The r_{alive} is a constant, which serves to ensure that the UAV earns a reward when flying within a defined region, in this case the r_{alive} = 1.5. The ϵ_t refers the distance between the target position and the UAV base at time step t, which can be seen by Equation 4.4.

$$\epsilon_t(s) = \sqrt{\xi^2_{target_{(t)}} - \xi^2_{uav_{(t)}}}$$

$$\epsilon_t(s) = \sqrt{(x_{target_{(t)}} - x_{uav_{(t)}})^2 - (y_{target_{(t)}} - y_{uav_{(t)}})^2 - (z_{target_{(t)}} - z_{uav_{(t)}})^2} \tag{4.4}$$

We added a cost for the absolute value of the relative angular velocities. We applied a higher penalty to the $\dot{\psi}$ since it was most responsible for the vibration (ringing effect) of our aircraft.

Note that since our r_{alive} = 1.5 and the horizontal time is 250, the maximum reward received can reach the value of 375, an important reference when discussing the results.

4.6.5 Episode completion

The agent resumes an episode and restarts another under the conditions listed below:

1. The was a collision;
2. The distance from target is greater than 19.5 meters;
3. The number of steps in an epoch exceed 250 time steps;
4. The UAV exitedd the defined global space.

4.6.6 Initialization

To initialize the UAV state at each episode, we used the *Discretized Uniform* initialization, proposed by [75].

I1: Initialization - Discretized Uniform

We defined a discrete uniform distribution in an array and that can parameterize how many pieces it would be divided. The dimension of the scenario was the parameter considered to define the size of distribution $num\_discretization$ and its limit $bound\_of\_distribution$, as shown below:

- For [x] $num\_discretization$ = 7 and $bound\_of\_distribution$ = [-3.000, 5.850]. Defining thus ([-3, -1.52, -0.05, 1.42, 2.9, 4.37, 5.85])
- For [y] $num\_discretization$ = 7 and $bound\_of\_distribution$ = [-2.125, 6.875]. Defining thus ([-2.12, -0.62, 0.88, 2.38, 3.88, 5.38, 6.88])
- For [z] $num\_discretization$ = 5 and $bound\_of\_distribution$ = [1, 2.5]. Defining thus ([1, 1.38, 1.75, 2.12, 2.5])
- For [ϕ, θ, ψ] $num\_discretization$ = 11 and $bound\_of\_distribution$ = [-0.785, 0.785]. Defining thus ([-0.78, -0.63, -0.47, -0.31, -0.16, 0, 0.16, 0.31, 0.47, 0.63, 0.78])

4.6.7 Action Space

The action space is composed by the actions of each propeller, defined from a PWM range from 0 to 100. This action space is given by A_p=$\{a_1, a_2, a_3, a_4\}$ and it is applied to each propeller through the Equation 4.1.

4.6.8 Algorithm Parameters

In this section, the settings of the SAC algorithm and Autoencoder will be presented.

Soft Actor-Critic (SAC)

The SAC algorithm settings follow the ones proposed by most open-source implementations, like [75]. However, some adaptations were necessary due to the significant increase of sensors, change of scenarios, increase in observation states, and complexity. The final hyper-parameters are listed in Table 4.3.

As we have several start points in our application, a good approach is to increase the batch size. Thus the task can be explored/evaluated from many configurations using the same trained policy π_t. These were the hyper-parameters that generated the best results so far.

Table 4.2 – Parameters - SAC Algorithm.

SAC Algorithm	
Parameter	**Value**
Batch size	4,000
Buffer size	5,000,000
Discount (γ)	0.99
Learning rate α	10^{-4}
Num train loops per step	1
Policy network	(64, $tanh$, 64, $tanh$)
Value and Soft-Q networks	(256, $relu$, 256, $relu$)

Autoencoder

We defined the autoencoder parameters from tests carried out directly in the scenes proposed in this work. The following parameters were considered the learning rate, network size, loss rate, batch size.

To reduce the computational cost of the algorithm, only four images were recorded, two current images and two previous images, seen by the floor and front cameras. Figures 4.9 and

4.10 represent the evolution of learning for some of these tests, in which our learning rate was defined as 0.001, batch size 4, and a maximum amount of 10,000 episodes.

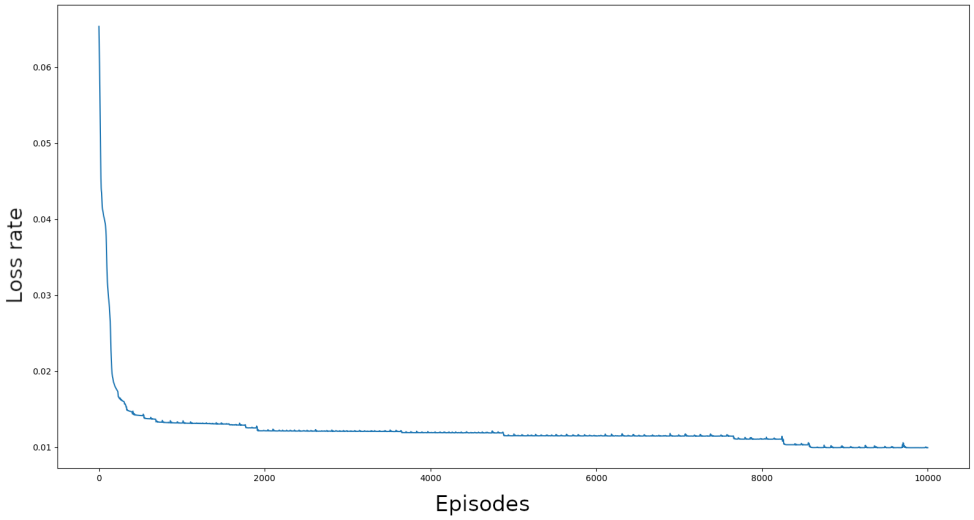


Figure 4.9 – AE Learning Curve - Assessment 1.

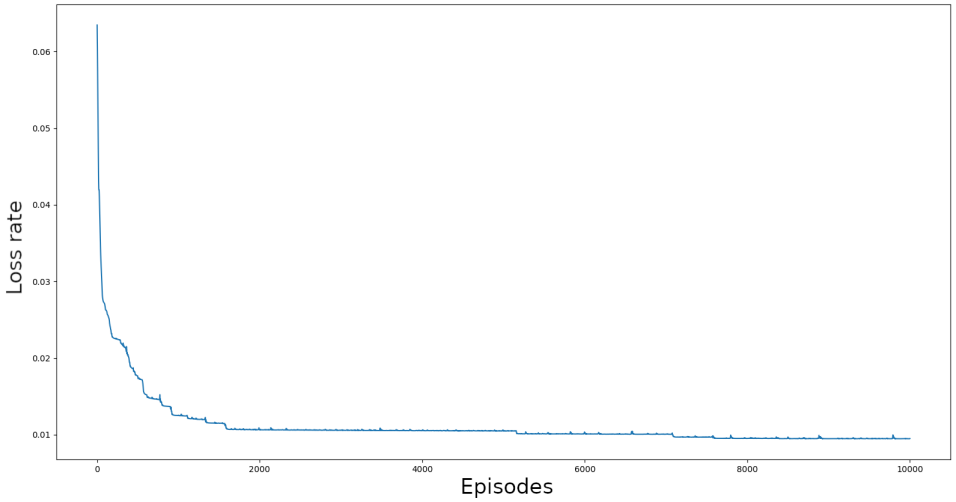


Figure 4.10 – AE Learning Curve - Assessment 2.

It can be seen in Figures 4.11, 4.12 and 4.13 the encoder (a) and decoder (b) of the networks right after training, using random images, but known by the network. With this, it was possible to achieve a decoder accuracy of 99.1%.

To validate the learning, we used a new database with 2.000 images from the same environment, not necessarily known by the network, so we selected 5 random images and verified the accuracy of the encode in these new images, which can be seen in the Figures 4.14, 4.15 and 4.16. We achieved an accuracy between 98.4% and 99.1%.

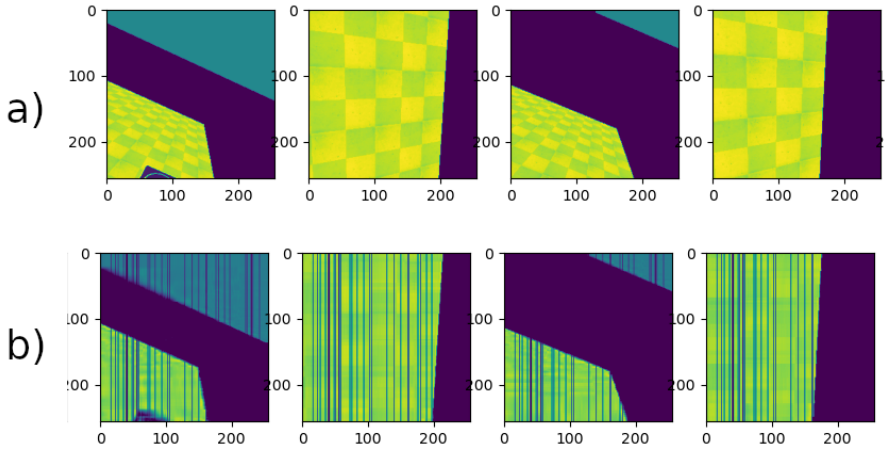


Figure 4.11 – AE Train - Assessment 1.

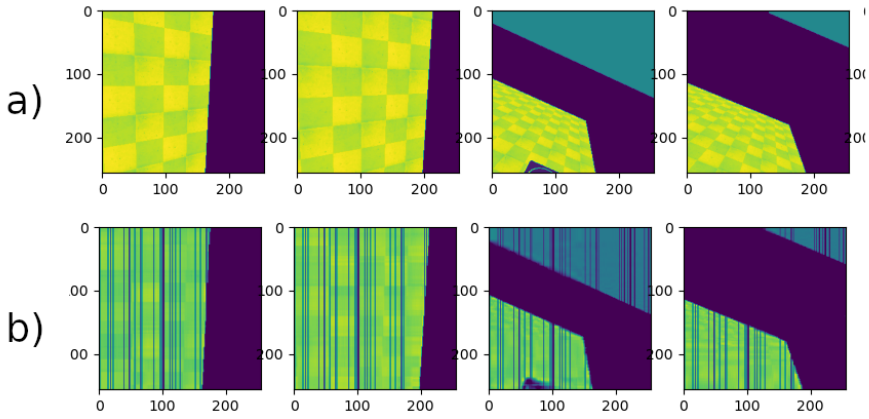


Figure 4.12 – AE Train - Assessment 2.

Thus, after several experiments, considering algorithm's precision and efficiency, the parameters that best met the expectations are defined in Table 4.3. Since some images, during testing, did not achieve the expected accuracy, each new batch of images will be forced to have an accuracy of 99.6% or a maximum value of 30 epochs of AE.

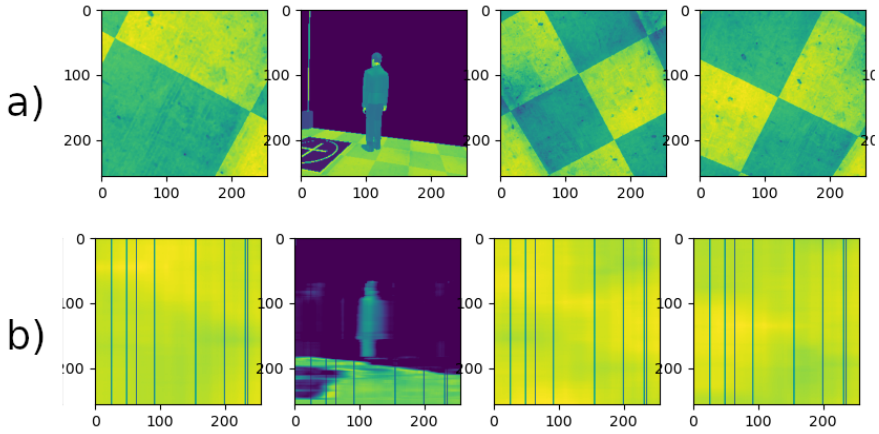


Figure 4.13 – AE Train - Assessment 3.

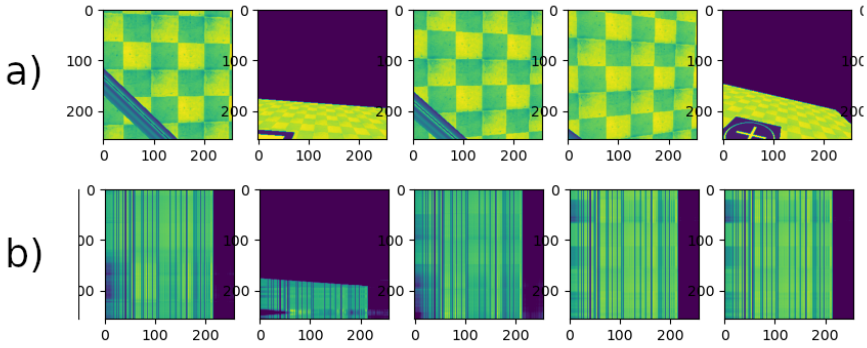


Figure 4.14 – AE Test - Assessment 1.

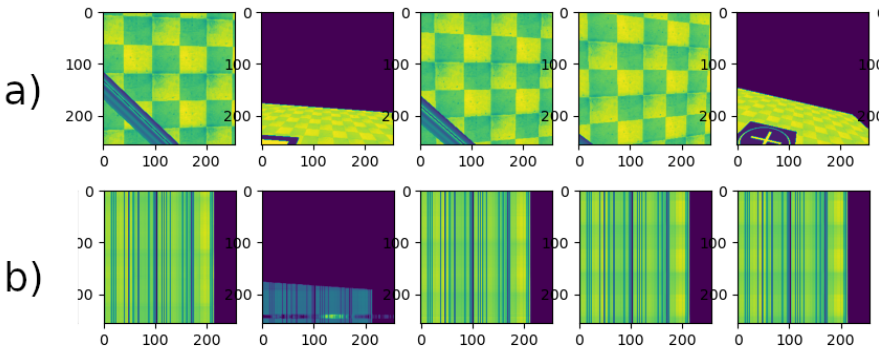


Figure 4.15 – AE Test - Assessment 2.

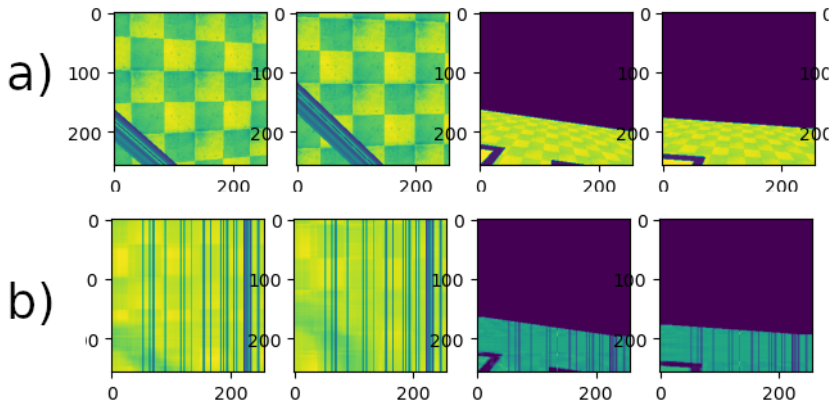


Figure 4.16 – AE Test - Assessment 3.

Table 4.3 – Parameters - Autoencoder Algorithm.

Autoencoder Algorithm	
Parameter	**Value**
Original image size	64x64
Image- Original/Converted	RGB / Grayscale
Batch size	4
Learning rate α	10^{-3}
Code networks	(32x32, $relu$, 16x16, $relu$, 8x8, $relu$, 4x4, $relu$, 2x2, $relu$)
Decode networks	(2x2, $relu$, 4x4, $relu$, 8x8, $relu$, 16x16, $relu$, 32x32, $relu$)
Loss rate	0.005
Max episodes	30

5 Results

In this chapter, we will present and discuss the results per scenario, assessing how the learning was affected per model proposed. We will discuss the influence of parameters, the resulting aircraft behavior, and the approaches used.

5.1 Approaches Overview

We tested several model configurations, parameters, reward strategies, observed states, and initialization strategies until the UAV reached a good behavior. We list some as:

1. First attempt, the aircraft should learn the stability and displacement in the environment simultaneously. Within the most challenging scenario - the SC3 Dynamic Obstacles. We used the same approach in the other scenarios but also without success.
2. Different terms in the reward function, as mentioned in Section 4.6.4.
3. Added different states like the target distance, global target position, and length timestep.
4. Fixed initialization to a specific global position and orientation.
5. Fixed initialization to a specific global position but with a variation in orientation.

All these approaches did not indicate a learning evolution. Therefore, we will not detail them further.

As a step-by-step approach proved to be more efficient in the learning process, we separated it into four steps. The scenario adopted in this first stage is SC0 - Empty Scenario, which previously already been performed successfully in [91] and [75]. In this step, we train the algorithm to stabilize the UAV in the empty scenario. We consider that by the end of this stage, the flight stability and accuracy have already reached an acceptable error rate, enabling a free displacement close to the ideal that can be verified by the SC1 - Free scenario. Thus, the expectation in the third stage is that the aircraft learns to avoid fixed obstacles for the scenario SC2 - Fixed Obstacle. Finally, in the last stage, the UAV is expected to learn to avoid dynamic obstacles. We will use the SC3 - Dynamic Obstacle scenario for this.

In order not to compromise UAV learning, the states will be partially enabled, evolving according to the stage. This evolution can be seen in table 5.1. More details will be presented in the section.

Table 5.1 – Sequence of enabled states.

Enabled States					
States	Unit	Empty Scenario	Free Scenario	Obstacle Scenario	Dynamic Scenario
UAV_Position_X_Y_Z	3	✓	✓	✓	✓
UAV_Rotation_Matrix	9	✓	✓	✓	✓
UAV_Angular_Velocity	3	✓	✓	✓	✓
UAV_Linear_Velocity	3	✓	✓	✓	✓
UAV_Propellers_Action	4	✓	✓	✓	✓
UAV_Ultrasonic_Sensors	10		✓	✓	✓
UAV_Global_Limit	1		✓	✓	✓
UAV_Travelled_Path	1			✓	✓
UAV_Last_Floor_Image	4			✓	
UAV_Last_Front_Image	4			✓	
UAV_Currently_Floor_Image	4			✓	
UAV_Currently_Front_Image	4			✓	
UAV_Autoencoder_Loss_Rate	1			✓	
UAV_Position_Env	3		✓	✓	✓

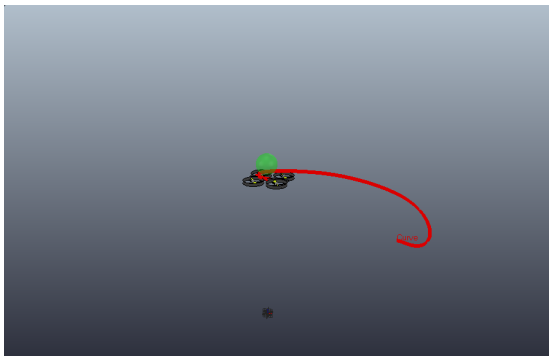
5.2 SC0 - Empty Scenario

In Figure 5.1, it is verified that the policy learned by the DRL was able to maximize the reward value, showing stability close to 125,000 timesteps, which represented 4,250 episodes which are approximated 3 days. To ensure the stability and accuracy of the UAV flight, the learned policy is submitted to the same scenario, but starting in different positions, global [*x, y, z*] and angular [ϕ, θ, ψ]. Tests performed demonstrated an average reward per step of 1.17, reaching a total reward of 293.7. Figure 5.2 shows the behavior of the UAV in a random test, showing the trajectory performed.

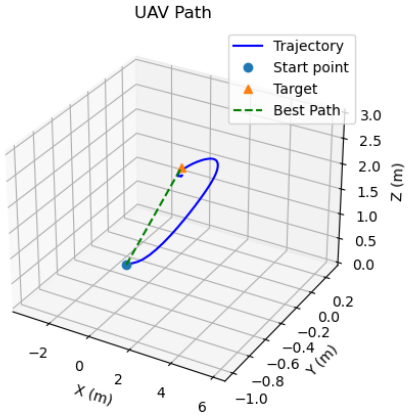
Note that despite the trajectory not being perfect, the aircraft maintained a behavior close to expected, achieving good accuracy in x and y and a small difference in z axis, verified by the graph - Final State Accuracy, seen in Figure 5.3. The UAV achieved a good dynamic movement quickly, attesting to a satisfactory degree of robustness. Still, it requires a little more precision due to the irregular amplitudes noted in the angular velocity curves, seen in Figure 5.4.

In this first stage with 4,250 epochs the UAV reached a satisfactory robustness degree. However, it is expected that with more training the DRL can achieve better accuracy, as achieved

Figure 5.1 – Average Reward - Epoch 4.250 - Empty scenario



(a) SC0 - Copppelia view



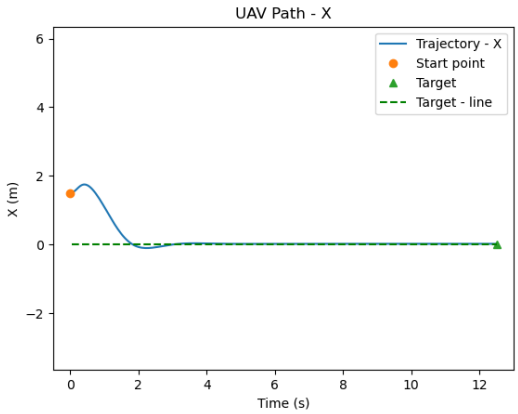
(b) SC0 - Cartesian plane

Figure 5.2 – SC0 - Path chosen by the UAV - Epoch 4.250 - Empty environment

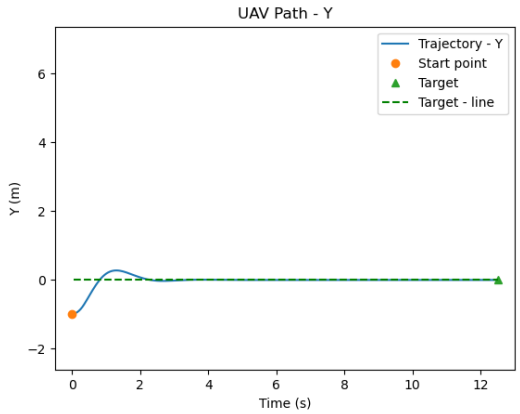
by [75]. However, this improvement can be achieved through training in the next scenarios, e.g. the SC1 - Free Scenario, which is the next to be explored. Thus, the policy learned in this stage will be transfer to the next and its new behavior will be checked.

5.3 SC1 - Free Scenario

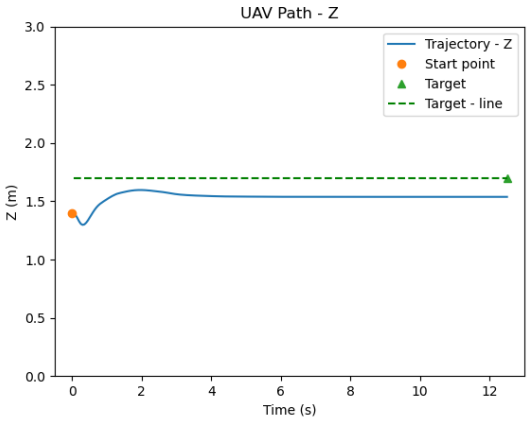
In this scenario, the UAV must adapt the learned policy to the addition in the states of ultrasonic sensors, as we insert new objects into the scene, previously unknown for the UAV. The global limit will be reduced on all axes, adapting to the arena's space.



(a) SC0 - x axis



(b) SC0 - y axis



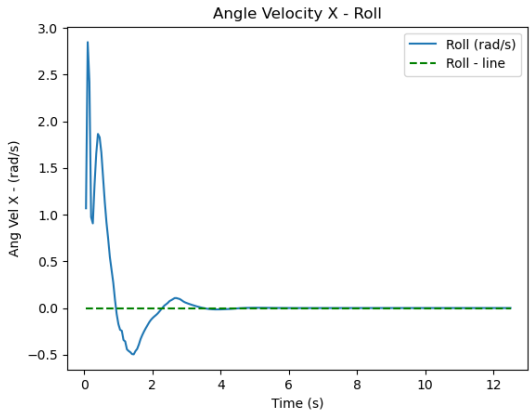
(c) SC0 - z axis

Figure 5.3 – SC0 - Final State Accuracy - $[x, y, z]$ axes

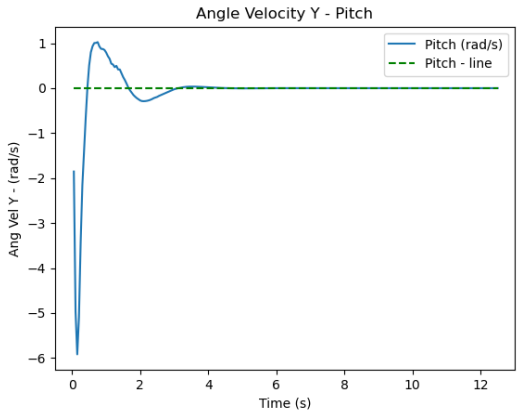
In this scenario, the aircraft must adapt to the new environment, fine-tuning the policy learned through previously unknown input variations. Since sudden variations in states can lead to inappropriate UAV behavior, including losing what has already been learned, we vary the states gradually, verifying if the learned behavior is performing as expected. At this stage, the UAV trained over 3,000 episodes, totaling 7,250 elapsed episodes.

The learning analysis follows the same methodology applied in the previous scenario. In Figure 5.5, we can see that the policy learned by the DRL enabled a dynamic behavior suitable for displacement within the free environment. The average rewards obtained in the tests were 0.706 per timestep and 176.52 per episode, which are good results considering the amount of additional training performed and the extra complexity of the environment.

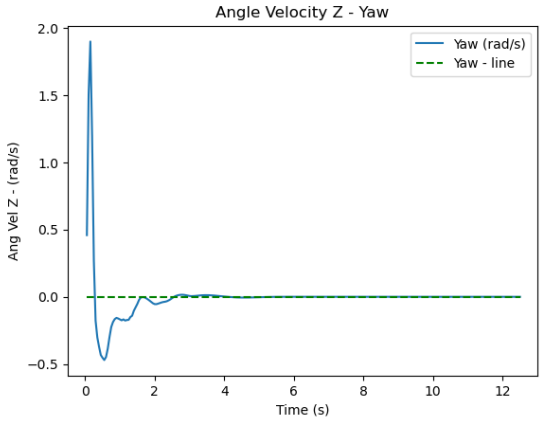
The view in the Cartesian plane of the path taken by the UAV can be analyzed in Figure 5.6. Although the path is not ideal, it reached 83.13% efficiency, comparing the distance covered and the shortest distance, which is certainly an encouraging value. The precision on the x and y axes was maintained, but the expected steady-state error reduction on the Z axis did not occur, which can be seen in Figure 5.7. In Figure 5.8, an apparent worsening of the angular velocity



(a) SC0 - Roll - $\dot{\phi}$



(b) SC0 - Pitch - $\dot{\theta}$

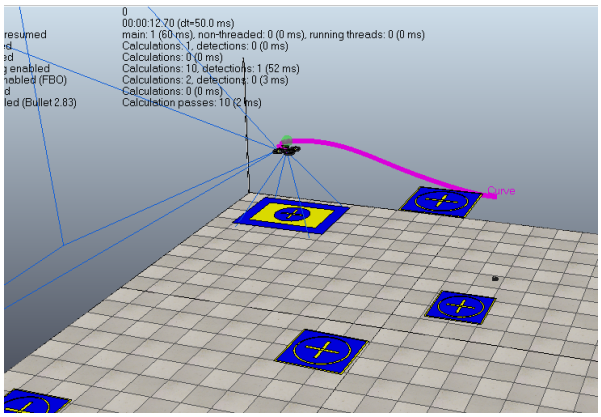


(c) SC0 - Yaw - $\dot{\psi}$

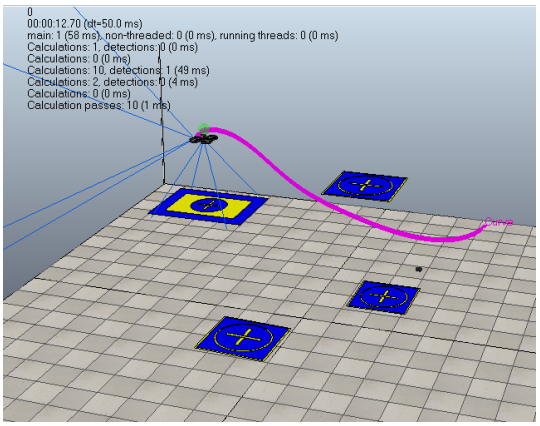
Figure 5.4 – SC0 - Angular Velocity - [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$] - Roll, Pitch, Yaw

behavior is shown, verified by the increase in the irregularities of the curve, but it is justified due to the increased complexity of the task and the fine-tuning performed by the network. Despite the apparent worsening, we have low steady-state error in the curves, which shows a robust UAV stability.

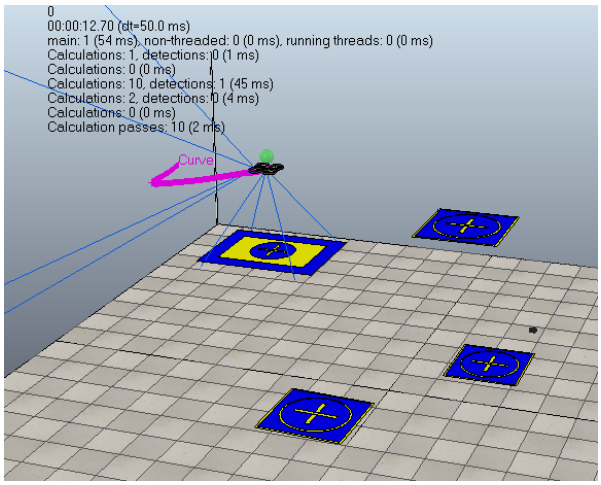
The behavior of the aircraft in this scenario shows a good evolution of the policy so that we will transfer it to the next stage.



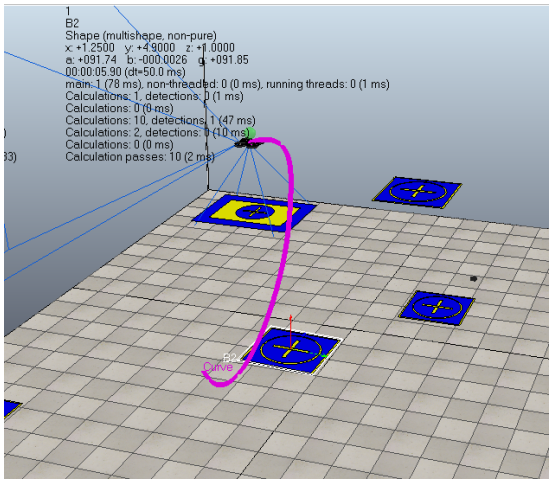
(a) SC1 - Test 1 - Coppelia view



(b) SC1 - Test 2 - Coppelia view



(c) SC1 - Test 3 - Coppelia view



(d) SC1 - Test 4 - Coppelia view

Figure 5.5 – SC1 - Path chosen by the UAV - Epoch 7,250 - Free environment

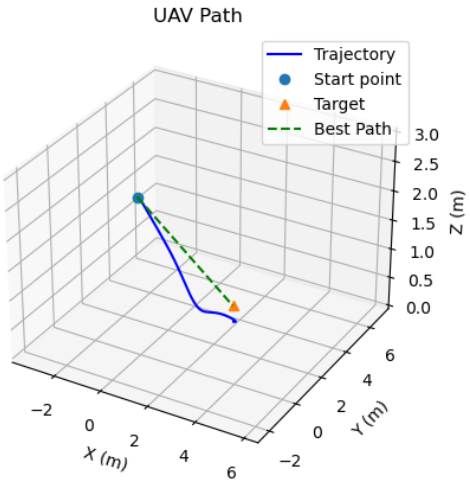
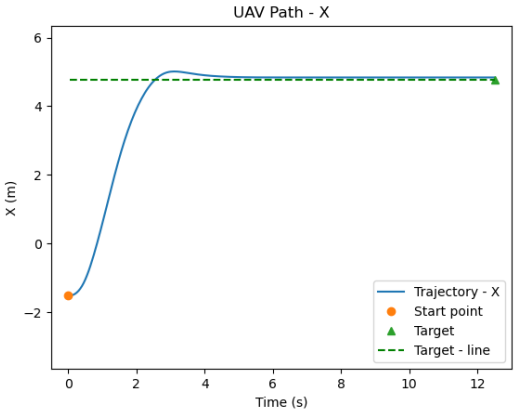
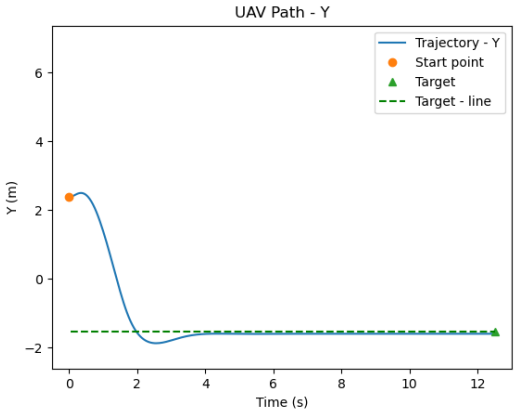


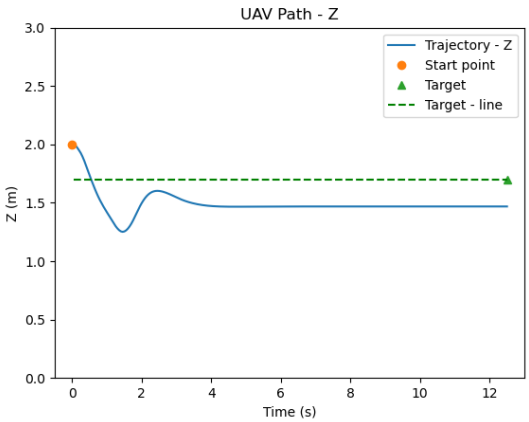
Figure 5.6 – SC1 - Cartesian plane - Path chosen by the UAV - Epoch 7.250 - Free Environment.



(a) SC1 - x axis

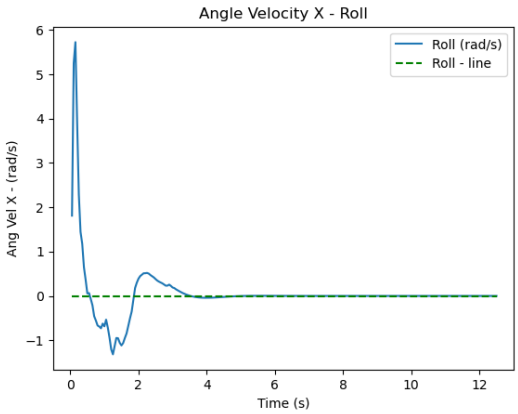


(b) SC1 - y axis

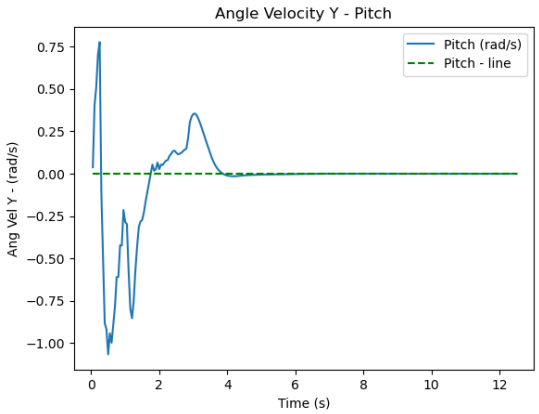


(c) SC1 - z axis

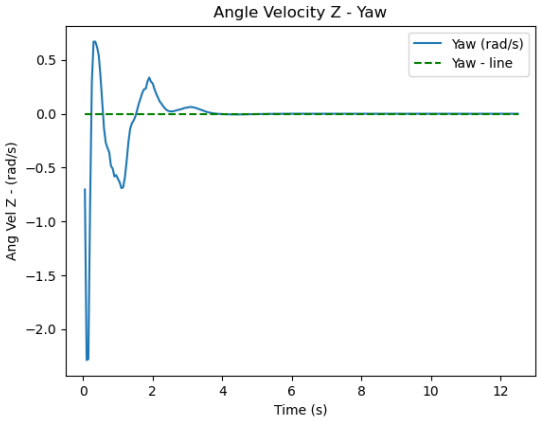
Figure 5.7 – SC1 - Final State Accuracy - $[x, y, z]$ axes



(a) SC1 - Roll - $\dot{\phi}$



(b) SC1 - Pitch - $\dot{\theta}$



(c) SC1 - Yaw - $\dot{\psi}$

Figure 5.8 – SC1 - Angular Velocity - [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$] - Roll, Pitch, Yaw

5.4 SC2 - Fixed Obstacles Scenario

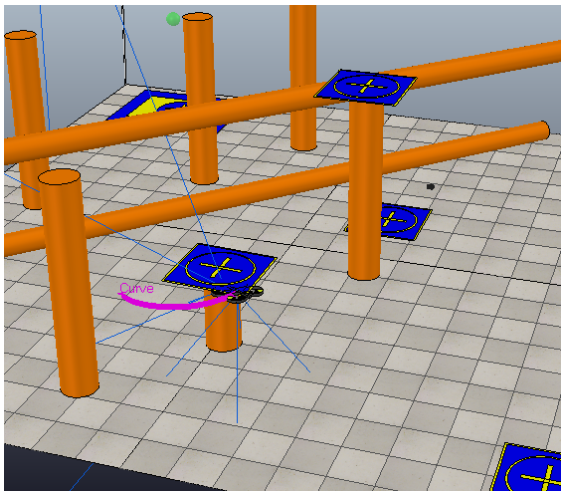
In this scenario, the UAV must learn to avoid the fixed obstacles in the environment. Considering that the UAV has already learned to move in an environment without obstacles, we believe that the aircraft will be able to recognize the obstacles around through visual and ultrasonic sensors, thus learning to maintain a safe and efficient route. Due to the greater complexity of the task, assessments will be gradual to validate the policy being seized and define whether we should continue with it.

We evaluated the best policy learned with 4,250, 7,250, 8,250, and 9,500 episodes. The resulting UAV dynamic behavior can be seen in Figure 5.9 and Figure5.11. The terms of comparison were the mean reward, sum reward, shortest path distance, the path traveled, and path efficiency. The terms evolution are detailed in Table 5.2.

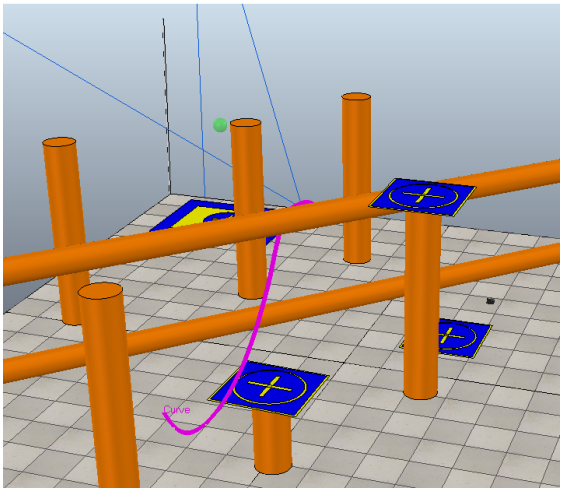
We can see an encouraging evolution in learning after train more than 2,500 episodes, totaling 9,500 elapsed episodes. Table 5.2 confirms the reward evolution in 5,250 episodes.

Table 5.2 – Learning Evolution - Fixed Obstacle Environment.

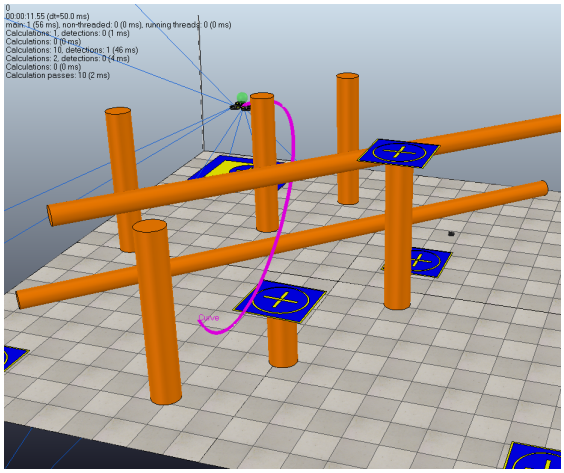
Learning Evolution - Obstacle				
Parameters	**4,250**	**7,250**	**8,250**	**9,500**
Mean Reward	-7.594	-4.005	0.212	0.255
Sum Reward	-113.91	-244.31	53.08	63.86
Best Path	1.82	9.89	9.10	9.06
Traveled Path	1.93	10.62	10.26	10.69
% Efficiency	94.30%	92.63%	87.27%	81.99%



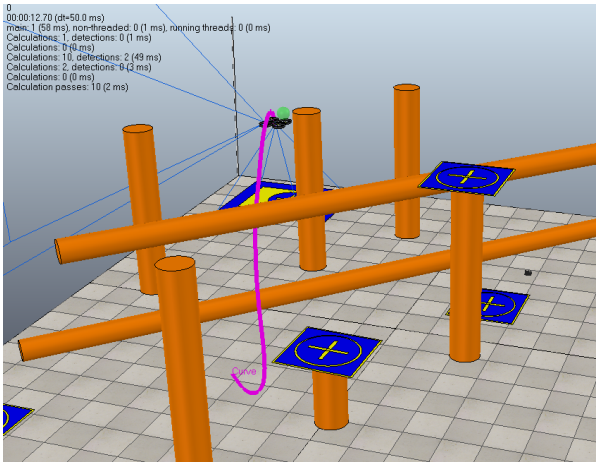
(a) SC2 - Epoch 4,250 - Coppelia view



(b) SC2 - Epoch 7,250 - Coppelia view



(c) SC2 - Epoch 8,250 - Coppelia view



(d) SC2 - Epoch 9,500 - Coppelia view

Figure 5.9 – SC2 - Learning Evolution - Epoch 9.500 - Fixed Obstacle environment

It is important to mention that the best path and traveled path presented in this table indicate the distance between the start and end points of the UAV, not the distance between the start point and the target. Figure 5.12 and Figure 5.13 show the path traveled by the UAV per axis and the angular velocities, respectively. Steady-state error in the z axis and abrupt variations in angular velocities persist, but it maintains good linear stability in the steady-state, showing good robustness.

As expected, every time new observation states are added, the network goes through an adaptation period, tending to converge again after this time. Figure 5.10 graphically represents this behavior, observe that there are 2 (two) breaking points that are moments that there was the addition of states, mentioned in Table 5.1. We can observe that during the change to Free-Scenario, the first breaking point, there was a drop in rewards but kept most of the behaviors considered good for the UAV. At the second breaking point, the switch to the Obstacle-Scenario, this drop in reward was very sharp. Hence, good behaviors that the agent had already learned were not maintained by the network, basically forcing a new training. In addition, the training time was approximately 30x longer than usual.

Due to the lack of time and hardware resources, the states of the visual sensors will be disabled in the dynamic environment, but the learning of other states will be transferred.

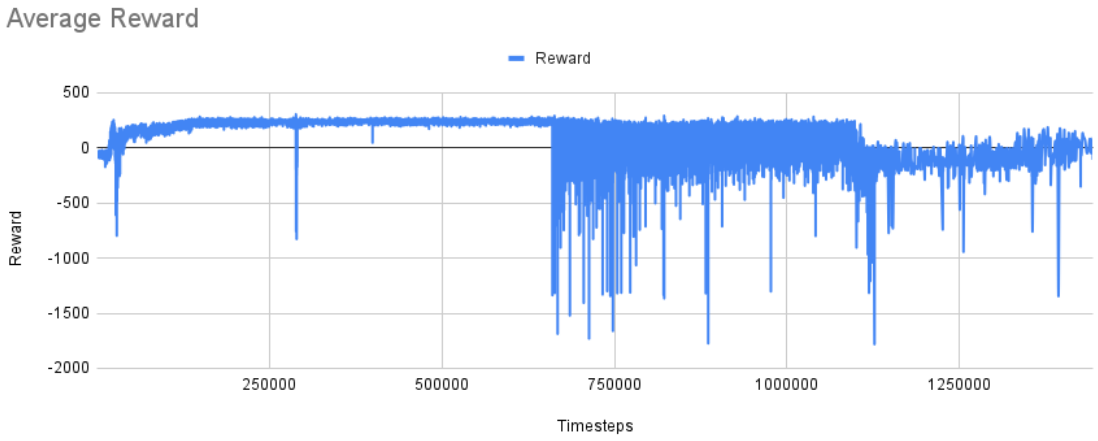
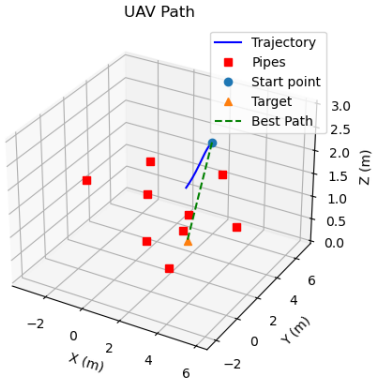
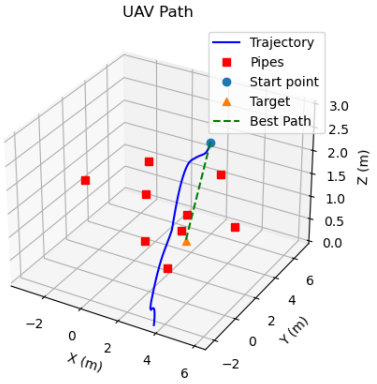


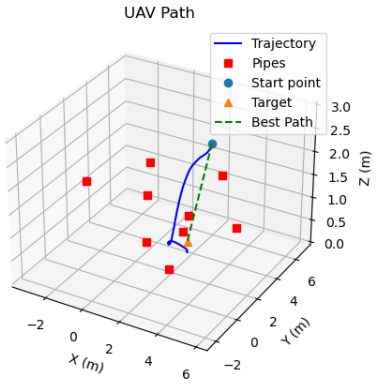
Figure 5.10 – Average Reward Evolution with the State Change



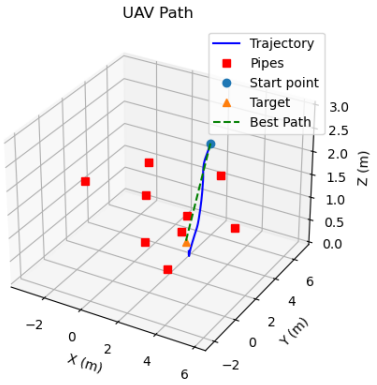
(a) SC2 - Epoch 4.250 - Cartesian Plane



(b) SC2 - Epoch 7.250 - Cartesian Plane

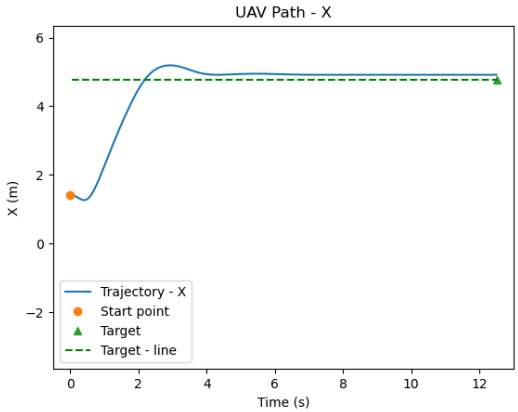


(c) SC2 - Epoch 8.250 - Cartesian Plane

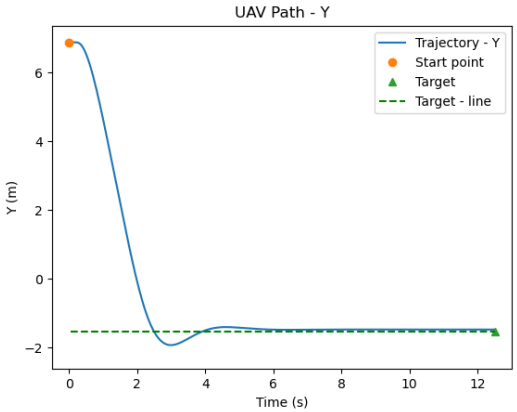


(d) SC2 - Epoch 9.500 - Cartesian Plane

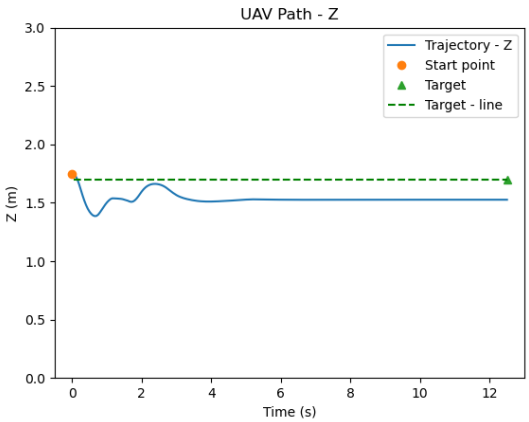
Figure 5.11 – SC2 - Cartesian Plane - Learning Evolution - Epoch 9.500 - Fixed Obstacle environment



(a) SC2 - x axis

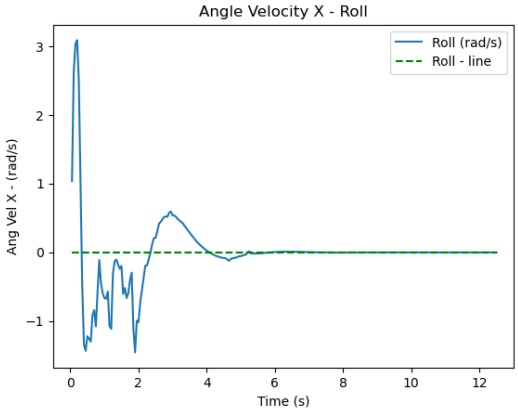


(b) SC2 - y axis

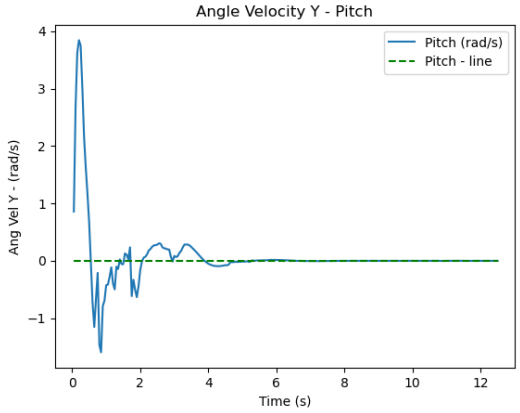


(c) SC2 - z axis

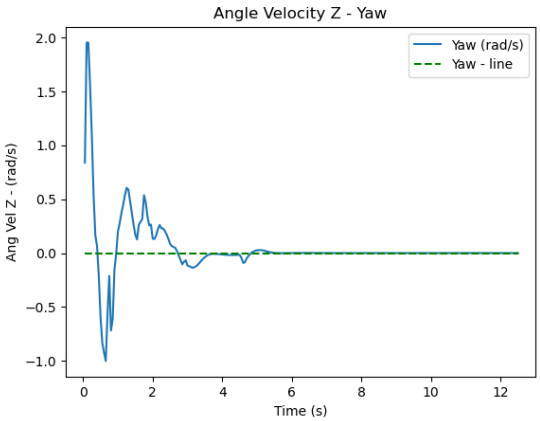
Figure 5.12 – SC2 - Final State Accuracy - $[x, y, z]$ axes



(a) SC2 - Roll - $\dot{\phi}$



(b) SC2 - Pitch - $\dot{\theta}$



(c) SC2 - Yaw - $\dot{\psi}$

Figure 5.13 – SC2 - Angular Velocity - [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$] - Roll, Pitch, Yaw

5.5 SC3 - Dynamic Obstacles Environment

For testing the UAV in a dynamic environment, we added six humanoids to the scene. The performance evaluation method will be the same as used in the previous ones. The policy learned will also be inherited and the evolution measured through comparisons at pre-defined time points, which are 10,500, 11,500, 12,500, and 13,500 episodes. We can observe the learning evolution in Table 5.3. Despite the collision in the second stage, there was a good evolution, and its successors are better than the others. In Figure 5.14 we show 4 (four) instants where the UAV was subject to a dynamic obstacle:

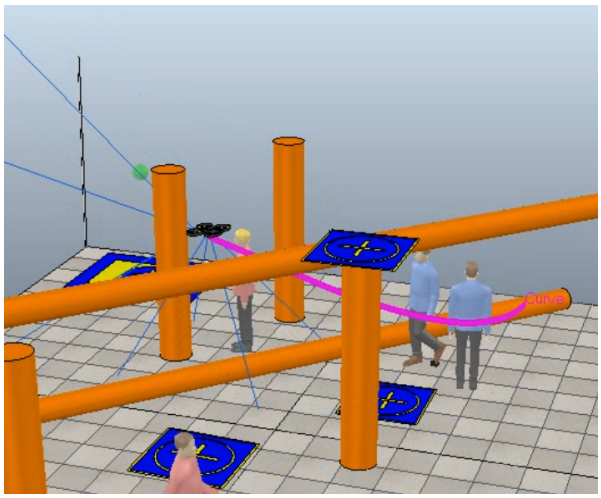
1. At the first moment in 10,500 episodes, the aircraft moves to the left and collides with a fixed obstacle;
2. In the second moment with 11,500 episodes, the UAV tries to execute a probably shorter path, but it hits the humanoid;
3. In the third attempt with 12,500 episodes, the UAV flies the humanoid's right and manages to reach the target without collision, showing a tendency to increase the flight height;
4. In the fourth attempt, with 13,500 episodes, the aircraft flies above the humanoid and goes back to the most central route, which would probably be the shortest.

Table 5.3 – Learning Evolution - Fixed Dynamic Environment.

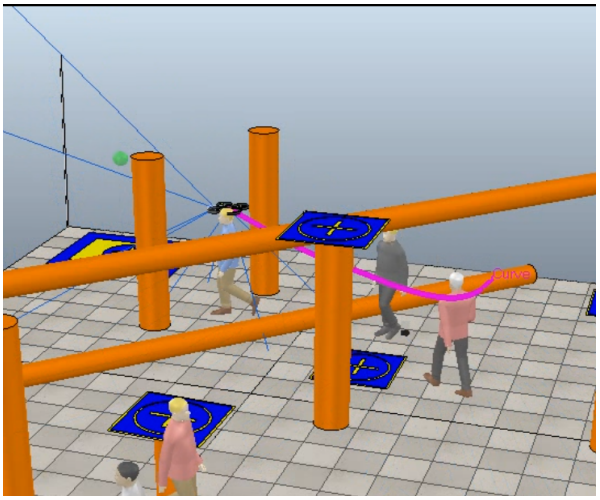
Learning Evolution - Dynamic				
Parameters	**10.500**	**11.500**	**12,500**	**13.500**
Mean Reward	0.394	-4.502	0.491	0.456
Sum Reward	98.609	-139.560	122.693	114.062
Best Path	7.27	4.47	7.37	7.41
Traveled Path	8.74	4.72	8.79	8.77
% Efficiency	81.32%	94.45%	80.75%	81.59%

In Figure 5.15 its seen the same path in a Cartesian plane. It is possible to observe that, as training evolves, the trajectory chosen by the UAV is increasingly closer to the ideal, tending to deviate only when the UAV find an obstacle. In Figure 5.15d, this evolution stands out, showing a sharp deviation when confront the mobile obstacle, however returning to the path later.

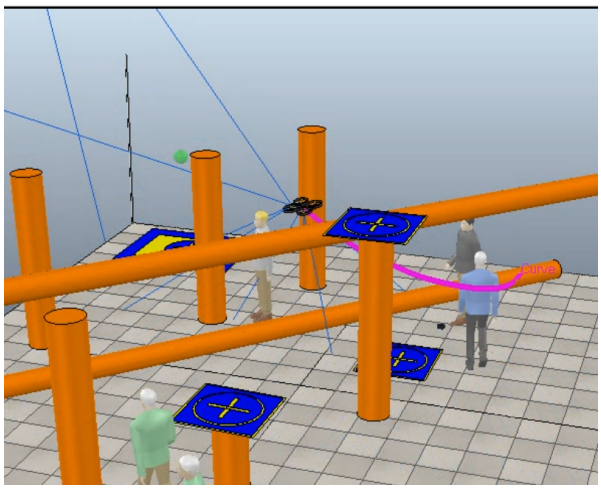
The position in time in each axis can be seen in Figure 5.16, the x and y axes continue with excellent precision. On the other hand, in the z axis, the steady-state error is persistent. In Figure 5.17 strong peaks in angular velocity change were observed, which despite not compromising the study, is likely to be refined as we evolve with training, as well as the error found in the z-axis.



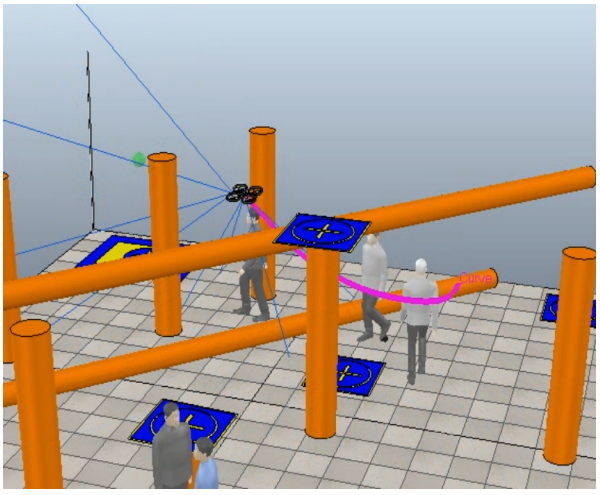
(a) SC3 - Epoch 10,500 - Coppelia view



(b) SC3 - Epoch 11,500 - Coppelia view

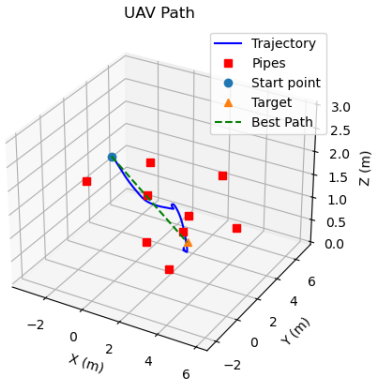


(c) SC3 - Epoch 12,500 - Coppelia view

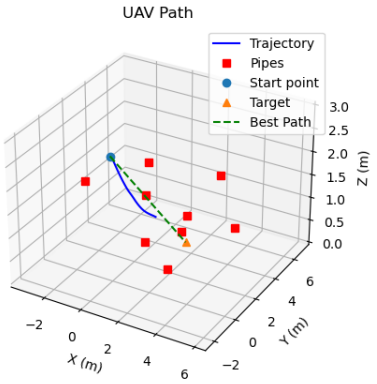


(d) SC3 - Epoch 13,500 - Coppelia view

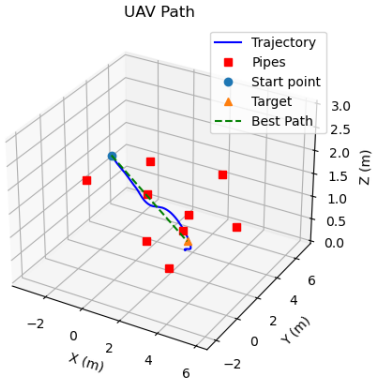
Figure 5.14 – SC3 - Learning Evolution - Epoch 13,500 - Dynamic environment



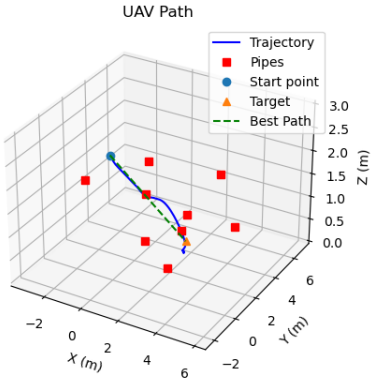
(a) SC3 - Epoch 10,500 - Cartesian Plane



(b) SC3 - Epoch 11,500 - Cartesian Plane

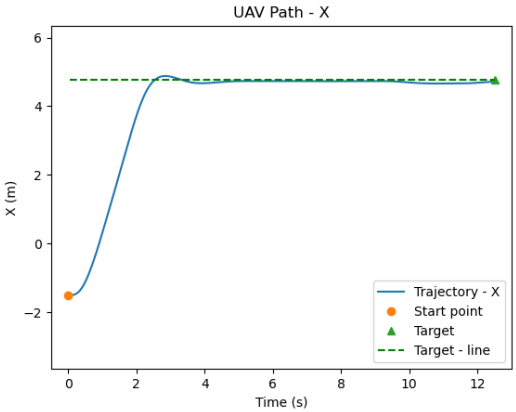


(c) SC3 - Epoch 12,500 - Cartesian Plane

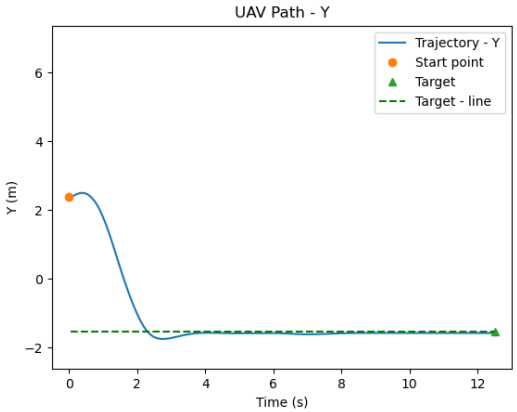


(d) SC3 - Epoch 13,500 - Cartesian Plane

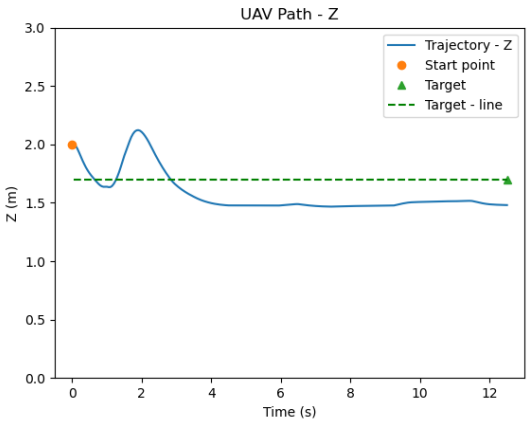
Figure 5.15 – SC3 - Cartesian Plane - Learning Evolution - Epoch 13,500 - Dynamic environment



(a) SC3 - x axis

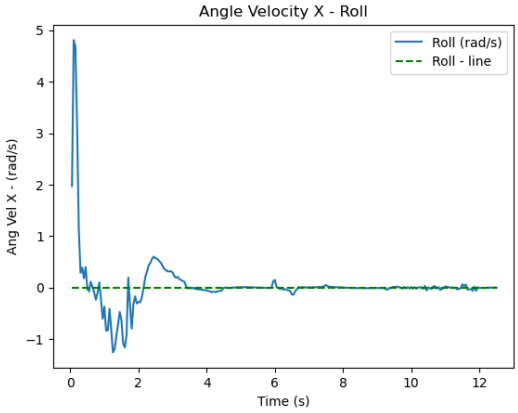


(b) SC3 - y axis

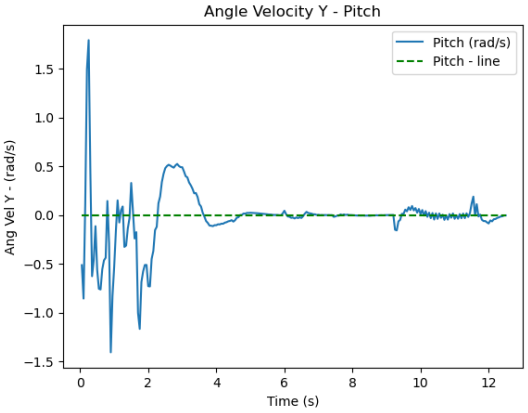


(c) SC3 - z axis

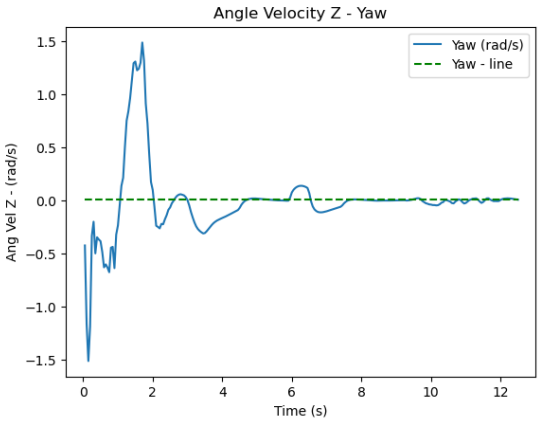
Figure 5.16 – SC3 - Final State Accuracy - $[x, y, z]$ axes



(a) SC3 - Roll - $\dot{\phi}$



(b) SC3 - Pitch - $\dot{\theta}$



(c) SC3 - Yaw - $\dot{\psi}$

Figure 5.17 – SC3 - Angular Velocity - [$\dot{\phi}$, $\dot{\theta}$, $\dot{\psi}$] - Roll, Pitch, Yaw

6 Conclusions and Future Works

In this work, we carried out an extensive study of the DRL techniques used for navigation and stability of UAVs in complex and dynamic environments. The approach chosen in this work was SAC, a state-of-art model-free off-policy algorithm based on maximum entropy RL that is very efficient in terms of search in the state-space.

Within the approaches tested in this work, we observed that the separation of the stability step and the navigation step was significant for the success of the learning process. SAC was able to perform low-level control of the UAV, corroborating the results obtained in [75] for a scenario without obstacles.

In [91] and [75], stability techniques for UAV were also addressed. In [75], SAC was also investigated, but the UAV was limited to track a moving target in an open environment. We investigated SAC's performance in an open environment and on environments with both fixed and mobile obstacles in the present work. With this approach, we aimed for a more realistic scenario. In this work, we built a new state space that includes information from vision and ultrasonic sensors, fundamental for identifying the obstacles. We employed a dimensionality reduction technique based on Autoencoder before inserting these sensor data into the state vector.

The current work also focused on investigating the generalization capability of the SAC algorithm during changes in the environment. The research demonstrated that it is possible to carry out autonomous and stable UAV navigation in free scenarios, with fixed and dynamic obstacles. This navigation typically requires some knowledge about the environment, which, in this case, was achieved through further training.

Due to lack of time and hardware resources, it was impossible to be conclusive about the benefits of including visual sensors in the UAV structure. However, our preliminary analysis point in the direction of a clear contribution to the aircraft flight in terms of navigation and stability.

Considering the research questions that guided this work, as presented in Section 1.1, we can state that:

1. **To investigate the representation of the states of UAVs in the DRL context, particularly focusing on the investigation of state representations that can simultaneously carry visual and other sensors information.**

   The states considered the structural dynamics of the aircraft, visual and ultrasonic sensors information, limits of the arena, the path traveled, autoencoder accuracy rate, and GNSS. The experiments with the state defined using these elements allowed a satisfactory precision and robust flight in all the proposed scenarios. We observed that the gradual increase in the complexity of the states had a significant impact on the UAV control.

2. **To investigate the aircraft navigability in environments with or without obstacles, fixed or in motion.**

   In this work was demonstrated that the SAC approach could perform low-level control of the UAV in all evaluated scenarios. Although good results have been obtained, the experiments suggest that the UAV can obtain a more accurate and robust flight with a more refined training process.

As future work, we suggest to:

- Investigate the effects of disturbances in the simulated environment and the evaluation of the behaviors learned by the UAV;
- Investigate approaches that can minimize UAV steady-state errors;
- Investigate the learning of the UAV in other environments and scenarios with increasing complexity (e.g., UAV swarm);
- Investigate strategies that can minimize the effects of the progressive inclusion of new states in the network.

Bibliography

1 Russell, S.; Norvig, P. *Artificial Intelligence - A modern Approach*. Third. [S.l.]: Pearson, 2010.

2 Dobitaobyte. *Deep Learning com Keras – Primeira rede neural*. 2020. Available from Internet: <https://www.dobitaobyte.com.br/deep-learning-com-keras-primeira-rede-neural/>.

3 Loey, M. *Figure uploaded by Mohamed Loey*. 2017. Available from Internet: <https://www.researchgate.net/figure/Sparse-autoencoder-structure_fig1_317734695>.

4 R. C. Gonzalez. Deep convolutional neural networks [lecture notes]. 2018. v. 35, n. 6, p. 79–87. *IEEE Signal Processing Magazine*.

5 Mitchell, T. *Machine Learning: A Guide to Current Research*. [S.l.: s.n.], 1997.

6 Alpaydin, E. *Introduction Machine Learning (third edition)*. Massachusetts Institute of Technology: [s.n.], 2014.

7 K. Arulkumaran, M. P. Deisenroth, M. Brundage e A. A. Bharath. Deep reinforcement learning: A brief survey. 2017. v. 34, n. 6, p. 26–38. *IEEE Signal Processing Magazine*.

8 Tang, H.; Haarnoja, T. *Learning diverse skills via maximum entropy deep reinforcement learning,*. 2017. <https://bair.berkeley.edu/blog/2017/10/06/soft-q-learning/>. Accessed: 2020-06-22.

9 Robotics, C. *Frameworks para V-REP*. Coppelia Robotics, 2019. Acesso em out. 2019. Available from Internet: <http://www.coppeliarobotics.com/helpFiles>.

10 PARROT. *Ardrone 2.0*. 2019. Available from Internet: <https://www.parrot.com/global/drones/parrot-ardrone-20-elite-edition3>.

11 M.Dorosti. Cooperative quadrotor formation flight in indoor environments robust min-max mpc approach,. 2011. v. 07. *Ph.D dissertation*.

12 Massey, N. *Humans May Be the Most Adaptive Species*. 2013. <https://www.scientificamerican.com/INPROCEEDINGS/humans-may-be-most-adaptive-species/>. Accessed: 2020-06-22.

13 G. Strawn. Alan turing. 2014. v. 16, n. 1, p. 5–7. *IT Professional*.

14 Kurzweil, R. *The Age of Intelligent Machines*. [S.l.]: MIT Press, 1990.

15 M. Xue e C. Zhu. A study and application on machine learning of artificial intellligence. *2009 International Joint Conference on Artificial Intelligence*. 2009. p. 272–274.

16 M. A. da Silva Ferreira, G. C. Lopes, E. L. Colombini e A. da Silva Simões. A novel architecture for multipurpose reconfigurable unmanned aerial vehicle (uav): Concept, design and prototype manufacturing. *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. 2018. p. 443–450. *doi:10.1109/LARS/SBR/WRE.2018.00085*.

17 C. Yang, T. Ma, Z. Zhang e G. Zhou. Research on structural and aeroelastic design of a hybrid multi-rotor uav. *2019 IEEE International Conference on Unmanned Systems (ICUS)*. 2019. p. 490–494. *doi:10.1109/ICUS48101.2019.8995940*.

18 A. Łukaszewicz, K. Szafran e J. Józwik. Cax techniques used in uav design process. *2020 IEEE 7th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. 2020. p. 95–98. *doi:10.1109/MetroAeroSpace48742.2020.9160091*.

19 N. Li, S. Yu e Z. Xi. Nonlinear control design for a quad rotor unmanned aerial vehicle. *2016 35th Chinese Control Conference (CCC)*. 2016. p. 469–474. *doi:10.1109/ChiCC.2016.7553129*.

20 B. Sielly Jales Costa, V. R. Greati, V. Campos Tinoco Ribeiro, C. S. da Silva e I. F. Vieira. A visual protocol for autonomous landing of unmanned aerial vehicles based on fuzzy matching and evolving clustering. *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2015. p. 1–6. *doi:10.1109/FUZZ-IEEE.2015.7337907*.

21 B. Vlahov, E. Squires, L. Strickland e C. Pippin. On developing a uav pursuit-evasion policy using reinforcement learning. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2018. p. 859–864. *doi:10.1109/ICMLA.2018.00138*.

22 A. Denuelle, R. Strydom e M. V. Srinivasan. Snapshot-based control of uas hover in outdoor environments. *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2015. p. 1278–1284. *doi:10.1109/ROBIO.2015.7418947*.

23 W. Wu, M. A. Qurishee, J. Owino, I. Fomunung, M. Onyango e B. Atolagbe. Coupling deep learning and uav for infrastructure condition assessment automation. *2018 IEEE International Smart Cities Conference (ISC2)*. 2018. p. 1–7. *doi:10.1109/ISC2.2018.8656971*.

24 Xiaodong Wang Ming Zhu Xiao Yang Liu. Deep reinforcement learning for unmanned aerial vehicle-assisted vehicular networks. 2019. *arXiv:1906.05015*.

25 H. Huang, Y. Yang, H. Wang, Z. Ding, H. Sari e F. Adachi. Deep reinforcement learning for uav navigation through massive mimo technique. 2020. v. 69, n. 1, p. 1117–1121. *IEEE Transactions on Vehicular Technology, doi:10.1109/TVT.2019.2952549*.

26 V. Saxena, J. Jaldén e H. Klessig. Optimal uav base station trajectories using flow-level models for reinforcement learning. 2019. v. 5, n. 4, p. 1101–1112. *IEEE Transactions on Cognitive Communications and Networking, doi:10.1109/TCCN.2019.2948324*.

27 B. H. Lee, J. R. Morrison e R. Sharma. Multi-uav control testbed for persistent uav presence: Ros gps waypoint tracking package and centralized task allocation capability. *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017. p. 1742–1750. *doi:10.1109/ICUAS.2017.7991424*.

28 C. Wang, J. Wang e X. Zhang. A deep reinforcement learning approach to flocking and navigation of uavs in large-scale complex environments. *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2018. p. 1228–1232. *doi:10.1109/GlobalSIP.2018.8646428*.

29 Walmart Apollo, Cantrell Robert, Alto Donald R, O'brien John J, Mchale Brian e Mattingly Todd. *Predictive Uav Package Delivery System*. 2018. Us201862624682p.

30 Amazon Tech Inc, Beckman Brian C[Us], Haskin Menashe[Il], Rolnik Michael[Il] e Vule Yan[Il] . *Maneuvering A Package Following In-Flight Release From An Unmanned Aerial Vehicle (Uav).* 2018. Us201815873354.

31 Walmart Apollo, High Donald R[Us], Cantrell Robert[Us] e Mchale Brian[Gb]. *Catch Nets Tower For Uav Delivery.* 2019. Us201916260752.

32 Walmart Apollo Llc, High Donald R[Us], Cantrell Robert[Us] e Mchale Brian[Gb]. *Temporary Uav Delivery Tower.* 2019. Us201916259445.

33 Walmart Apollo Llc, High Donald R[Us], Cantrell Robert[Us] e Mchale Brian[Gb]. *Retractable Table Tower For Uav Package Delivery.* 2019. Us201916260683.

34 Walmart Apollo e Cantrell Robert. *Laser-Guided Uav Delivery System.* 2017. Us201762459673p.

35 Intel Corp. Lamkin Andrew F e Wong Hong W. *Emergency Uav Method And Apparatus.* 2016. Us201616307373.

36 K. Yoon, D. Park, Y. Yim, K. Kim, S. K. Yang e M. Robinson. Security authentication system using encrypted channel on uav network. *2017 First IEEE International Conference on Robotic Computing (IRC)*. 2017. p. 393–398.

37 S. ur Rahman, G. Kim, Y. Cho e A. Khan. Positioning of uavs for throughput maximization in software-defined disaster area uav communication networks. 2018. v. 20, n. 5, p. 452–463. *Journal of Communications and Networks*.

38 H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah e M. Guizani. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. 2019. v. 7, p. 48572–48634. *IEEE Access*.

39 PWC. *PwC. Global Market for Commercial Applications of Drone Technology Valued at Over 127bn.* 2016. <https://pwc.blogs.com/press_room/2016/05/global-market-for-commercial-applications-of-drone-technology-valued-at-over-127bn.html>. Accessed: 2020-06-23.

40 T. Kelly. *The Booming Demand for Commercial Drone Pilots.* 2017. <https://www.theatlantic.com/technology/archive/2017/01/drone-pilot-school/515022/>. Accessed: 2020-06-23.

41 Joshi, D. *Commercial Unmanned Aerial Vehicle (UAV) Market Analysis,Industry Trends, Companies and What You Should Know.* 2017. <http://www.businessinsider.com/commercial-uav-market-analysis-2017-8>. Accessed: 2020-06-23.

42 B. Zhang, C. H. Liu, J. Tang, Z. Xu, J. Ma e W. Wang. Learning-based energy-efficient data collection by unmanned vehicles in smart cities. 2018. v. 14, n. 4, p. 1666–1676. *IEEE Transactions on Industrial Informatics*.

43 Y. Choi, H. Jimenez e D. N. Mavris. Two-layer obstacle collision avoidance with machine learning for more energy-efficient unmanned aircraft trajectories. 2017. v. 98, n. doi:10.1016/j.robot.2017.09.004. *Robot.Auto.System*.

44 Omar Bouhamed, Hakim Ghazzi, Hichem Besbes e Yehia Massoud. Autonomous uav navigation: A ddpg-based deep reinforcement learning approach. *IEEE International Symposium on Circuits and Systems (ISCAS'20)*. 2020.

45 M. Mamdouh, M. A. I. Elrukhsi e A. Khattab. Securing the internet of things and wireless sensor networks via machine learning: A survey. *2018 International Conference on Computer and Applications (ICCA)*. 2018. p. 215–218.

46 V. Lourenço, P. Mann, A. Guimarães, A. Paes e D. de Oliveira. Towards safer (smart) cities: Discovering urban crime patterns using logic-based relational machine learning. *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018. p. 1–8.

47 Charniak, E.; McDermott, D. *Introduction to Artificial Intelligence*. Addison-Wesley: [s.n.], 1985.

48 Winston, P. H. *Artificial Intelligence (Third edition)*. Addison-Wesley: [s.n.], 1992.

49 Poole D., M. A. K.; Goebel, R. *Computational intelligence: A logical approach*. Oxford University Press: [s.n.], 1998.

50 Nilsson, N. J. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann: [s.n.], 1998.

51 P. Louridas e C. Ebert. Machine learning. 2016. v. 33, n. 5, p. 110–115. *IEEE Software*.

52 Alexandre da Silva Simoes. Document class. 2020. Available from Internet: <http://lattes.cnpq.br/1368002066043197>.

53 Bengio, Y. *Learning Deep Architectures for AI*. Foundations and Trends in Machine Learning: [s.n.], 2009. 1–127. p.

54 Brownlee, J. *How to Code a Neural Network with Backpropagation In Python (from scratch)*. 2019. Available from Internet: <https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>.

55 D. E. Rumelhart e J. L. McClelland. Learning internal representations by error propagation. ____. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. [S.l.: s.n.], 1987. p. 318–362.

56 Werbos P.J. Beyond regression: New tools for prediction and analysis in the behavioral sciences. *Doctoral Dissertation Harvard University, Cambridge.* 1974. p. 318–362.

57 Volodymyr Kuleshov e Stefano Ermon. Deep hybrid models: Bridging discriminative and generative approaches. 2017. Available from Internet: <http://ai.stanford.edu/~ermon/papers/uai2017_cr.pdf>. *ai.stanford.edu*.

58 roboticsbiz. *Different types of Deep Learning models explained*. 2020. Available from Internet: <https://roboticsbiz.com/different-types-of-deep-learning-models-explained/>.

59 P. Munro Cottrell G. W. e D. Zipser. Learning internal representations from gray-scale images: An example of extensional programming. *Ninth Annual Conference of the Cognitive Science Society*. 1987. p. 462–473.

60 I. Arel, D. C. Rose e T. P. Karnowski. Deep machine learning - a new frontier in artificial intelligence research [research frontier]. 2010. v. 5, n. 4, p. 13–18. *IEEE Computational Intelligence Magazine*.

61 E. Shelhamer, J. Long e T. Darrell. Fully convolutional networks for semantic segmentation. 2017. v. 39, n. 4, p. 640–651. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

62 S. Lange, M. Riedmiller e A. Voigtländer. Autonomous reinforcement learning on raw visual input data in a real world application. *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 2012. p. 1–8.

63 Ioannis Antonoglou David Silver Tom Schaul John Quan. Prioritized experience replay. 2016. *ICLR*.

64 Matteo Hessel Hado van Hasselt Marc Lanctot Nando de Freitas Ziyu Wang Tom Schaul. Dueling network architectures for deep reinforcement learning. 2016. *ICLR*.

65 Ilya Sutskever Sergey Levine Shixiang Gu T. Lillicrap. Continuous deep q-learning with model-based acceleration. 2016. *ICLR*.

66 Jürgen Schmidhuber Faustino Gomez Jan Koutník Giuseppe Cuccu. Evolving large-scale neural networks for vision-based reinforcement learning. 2013. *Proc. Conf. Genetic and Evolutionary Computation*.

67 Philipp Moritz Michael Jordan Pieter Abbeel John Schulman Sergey Levine. Trust region policy optimization. 2015. *International Conference on Machine Learning*.

68 Sergey Levine Michael I. Jordan John Schulman Philipp Moritz e Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. 2016. *Conference paper at ICLR 2016*.

69 Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei e A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017. p. 3357–3364.

70 Pieter Abbeel Sergey Levine Tuomas Haarnoja A. Zhou. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv:1801.01290v2*. 2018.

71 Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. *Carnegie Mellon University,*. 2010.

72 B. D. Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

73 SPINNINGUP. *Soft Actor-Critic*. 2018. Available from Internet: <https://spinningup.openai.com/en/latest/algorithms/sac.html>.

74 Pieter Abbeel Tuomas Haarnoja Haoran Tang e Sergey Levine. Reinforcement learning with deep energy-based policies. *Proceedings of the 34 th International Conference on Machine Learning, Sydney, Australia*. 2017.

75 Barros, G. M. *Reinforcement and Imitation Learning Applied to Autonomous Aerial Robot Control*. 2020. <http://www.repositorio.unicamp.br/bitstream/REPOSIP/345879/1/Barros_GabrielMoraes_M.pdf>. Accessed: 2020-11-08.

76 A. Hernandez, C. Copot, R. De Keyser, T. Vlas e I. Nascu. Identification and path following control of an ar.drone quadrotor. *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)*. 2013. p. 583–588. *doi:10.1109/ICSTCC.2013.6689022*.

77 V. Kharchenko, D. Matiychyk e A. Babenko. Mathematical model of unmanned aerial vehicle control in manual or semiautomatic modes. *2017 IEEE 4th International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*. 2017. p. 37–40. *doi:10.1109/APUAVD.2017.8308771*.

78 D. H. C. Silva, M. F. Santos, M. F. Silva, A. F. S. Neto e P. Mercorelli. Design of controllers applied to autonomous unmanned aerial vehicles using software in the loop. *2019 20th International Carpathian Control Conference (ICCC)*. 2019. p. 1–6. *doi:10.1109/CarpathianCC.2019.8766036*.

79 S. Kim, V. Deshpande e R. Bhattacharya. 2 optimized pid control of quad-copter platform with wind disturbance. *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020. p. 839–844. *doi:10.1109/ICUAS48674.2020.9214010*.

80 M. A. Al-Shabi, K. S. Hatamleh e A. A. Asad. Uav dynamics model parameters estimation techniques: A comparison study. *2013 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT)*. 2013. p. 1–6. *doi:10.1109/AEECT.2013.6716436*.

81 Z. Birnbaum, A. Dolgikh, V. Skormin, E. O'Brien e D. Muller. Unmanned aerial vehicle security using recursive parameter estimation. *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014. p. 692–702. *doi:10.1109/ICUAS.2014.6842314*.

82 G. Tartaglione e M. Ariola. Development of an autonomous multi-rotor uav for outdoor missions in unknown environments. *2017 25th Mediterranean Conference on Control and Automation (MED)*. 2017. p. 1017–1022. *doi:10.1109/MED.2017.7984251*.

83 D. Y. Dube e R. K. Munje. Modeling and control of unmanned aerial vehicle. *2015 International Conference on Energy Systems and Applications*. 2015. p. 641–644. *doi:10.1109/ICESA.2015.7503428*.

84 Sachin Verma Pankaj Kumar Sa Sambit Bakshi Ram Prasad Padhy Shahzad Ahmad. Localization of unmanned aerial vehicles in corridor environments using deep learning. 2019. *arXiv:1903.09021*.

85 Roland Siegwart Marco Hutter Jemin Hwangbo Inkyu Sa. Control of a quadrotor with reinforcement learning. 2017. *arXiv:1707.05110*.

86 T. Sugimoto e M. Gouko. Acquisition of hovering by actual uav using reinforcement learning. *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*. 2016. p. 148–152. *doi:10.1109/ICISCE.2016.42*.

87 R. Sharma. Fuzzy q learning based uav autopilot. *2014 Innovative Applications of Computational Intelligence on Power, Energy and Controls with their impact on Humanity (CIPECH)*. 2014. p. 29–33. *doi:10.1109/CIPECH.2014.7019067*.

88 Wolfgang H¨onig James A. Preiss Nora Ayanian Artem Molchanov Tao Chen e Gaurav S. Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. 2019. *arXiv:1903.04628v2*.

89 Jie Xu, Tao Du, Michael Foshey, Beichen Li, Bo Zhu, Adriana Schulz e Wojciech Matusik. Learning to fly: Computational controller design for hybrid uavs with reinforcement learning. 2019. v. 38, n. 4. ISSN 0730-0301. Available from Internet: <https://doi.org/10.1145/3306346.3322940>. *ACM Trans. Graph., doi:10.1145/3306346.3322940*.

90 A. Villanueva e A. Fajardo. Uav navigation system with obstacle detection using deep reinforcement learning with noise injection. *2019 International Conference on ICT for Smart Society (ICISS)*. 2019. v. 7, p. 1–6. *doi:10.1109/ICISS48059.2019.8969798*.

91 G. Cano Lopes, M. Ferreira, A. da Silva Simões e E. Luna Colombini. Intelligent control of a quadrotor with proximal policy optimization reinforcement learning. *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. 2018. p. 503–508. *doi:10.1109/LARS/SBR/WRE.2018.00094*.

92 J.Schulman, F.Wolski, P.Dhariwal, A.Radford, e O.Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 2017.

93 Azer Bestavros William Koch Renato Mancuso. Neuroflight: Next generation flight control firmware. 2019. *arXiv:1901.06553v2*.

94 B. G. Maciel-Pearson, S. Akçay, A. Atapour-Abarghouei, C. Holder e T. P. Breckon. Multi-task regression-based learning for autonomous unmanned aerial vehicle flight control within unstructured outdoor environments. 2019. v. 4, n. 4, p. 4116–4123. *doi:10.1109/LRA.2019.2930496*.

95 J. Xu, Q. Guo, L. Xiao, Z. Li e G. Zhang. Autonomous decision-making method for combat mission of uav based on deep reinforcement learning. *2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. 2019. v. 1, p. 538–544. *doi:10.1109/IAEAC47372.2019.8998066*.

96 S. Cho, D. H. Kim e Y. W. Park. Learning drone-control actions in surveillance videos. *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. 2017. p. 700–703. *doi:10.23919/ICCAS.2017.8204319*.

97 O. Bouhamed, H. Ghazzai, H. Besbes e Y. Massoud. Autonomous uav navigation: A ddpg-based deep reinforcement learning approach. *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2020. p. 1–5. *doi:10.1109/ISCAS45731.2020.9181245*.

98 T.P.Lillicrap, J.J.Hunt, A.Pritzel, N.Heess, T.Erez, D.Silver Y.Tassa e D.Wierstra. Continuous control with deep reinforcement learning. *ICLR, 2016*. 2016.

99 Chao Xu Xin Zhou Zhepei Wang e Fei Gao. Ego-planner: An esdf-free gradient-based local planner for quadrotors. 2020. *arXiv:2008.08835*.

100 S. Grzonka, G. Grisetti e W. Burgard. A fully autonomous indoor quadrotor. 2012. v. 28, n. 1, p. 90–100. *IEEE Transactions on Robotics, doi:10.1109/TRO.2011.2162999*.

101 A. Annaiyan, M. A. Olivares-Mendez e H. Voos. Real-time graph-based slam in unknown environments using a small uav. *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2017. p. 1118–1123. *doi:10.1109/ICUAS.2017.7991524*.

102 J. Q. Cui, S. Lai, X. Dong, P. Liu, B. M. Chen e T. H. Lee. Autonomous navigation of uav in forest. *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2014. p. 726–733. *doi:10.1109/ICUAS.2014.6842317*.

103 Y. Hsu e R. Gau. Reinforcement learning-based collision avoidance and optimal trajectory planning in uav communication networks. 2020. p. 1–1. *IEEE Transactions on Mobile Computing, doi:10.1109/TMC.2020.3003639*.

104 C. Wang, J. Wang, X. Zhang e X. Zhang. Autonomous navigation of uav in large-scale unknown complex environment with deep reinforcement learning. *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2017. p. 858–862. *doi:10.1109/GlobalSIP.2017.8309082*.

105 C. Yan e X. Xiang. A path planning algorithm for uav based on improved q-learning. *2018 2nd International Conference on Robotics and Automation Sciences (ICRAS)*. 2018. p. 1–5. *doi:10.1109/ICRAS.2018.8443226*.

106 V. Hoang, D. Seo, L. Kurnianggoro e K. Jo. Path planning and global trajectory tracking control assistance to autonomous vehicle. *2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2014. p. 646–650. *doi:10.1109/URAI.2014.7057486*.

107 C. Wu, B. Ju, Y. Wu, X. Lin, N. Xiong, G. Xu, H. Li e X. Liang. Uav autonomous target search based on deep reinforcement learning in complex disaster scene. 2019. v. 7, p. 117227–117245. *IEEE Access, doi:10.1109/ACCESS.2019.2933002*.

108 S. Chen, S. Guo e Y. Li. Real-time tracking a ground moving target in complex indoor and outdoor environments with uav. *2016 IEEE International Conference on Information and Automation (ICIA)*. 2016. p. 362–367. *doi:10.1109/ICInfA.2016.7831851*.

109 Y. Liu, Q. Wang, H. Hu e Y. He. A novel real-time moving target tracking and path planning system for a quadrotor uav in unknown unstructured outdoor scenes. 2019. v. 49, n. 11, p. 2362–2372. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, doi:10.1109/TSMC.2018.2808471*.

110 W. Zhang, K. Song, X. Rong e Y. Li. Coarse-to-fine uav target tracking with deep reinforcement learning. 2019. v. 16, n. 4, p. 1522–1530. *IEEE Transactions on Automation Science and Engineering, doi:10.1109/TASE.2018.2877499*.

111 D. Ebrahimi, S. Sharafeddine, P. Ho e C. Assi. Autonomous uav trajectory for localizing ground objects: A reinforcement learning approach. 2020. p. 1–1. *IEEE Transactions on Mobile Computing, doi:10.1109/TMC.2020.2966989*.

112 S. Ku, S. Jung e C. Lee. Uav trajectory design based on reinforcement learning for wireless power transfer. *2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. 2019. p. 1–3. *doi:10.1109/ITC-CSCC.2019.8793294*.

113 Y. Lin, M. Wang, X. Zhou, G. Ding e S. Mao. Dynamic spectrum interaction of uav flight formation communication with priority: A deep reinforcement learning approach. 2020. v. 6, n. 3, p. 892–903. *IEEE Transactions on Cognitive Communications and Networking, doi:10.1109/TCCN.2020.2973376*.

114 D. Kwon e J. Kim. Optimal trajectory learning for uav-bs video provisioning system: A deep reinforcement learning approach. *2019 International Conference on Information Networking (ICOIN)*. 2019. p. 372–374. *doi:10.1109/ICOIN.2019.8718194*.

115 S. Yin, S. Zhao, Y. Zhao e F. R. Yu. Intelligent trajectory design in uav-aided communications with reinforcement learning. 2019. v. 68, n. 8, p. 8227–8231. *doi:10.1109/TVT.2019.2923214*.

116 H. Huang, Y. Yang, H. Wang, Z. Ding, H. Sari e F. Adachi. Deep reinforcement learning for uav navigation through massive mimo technique. 2020. v. 69, n. 1, p. 1117–1121. *IEEE Transactions on Vehicular Technology, doi:10.1109/TVT.2019.2952549*.

117 Min Liu Bo Yang. Keeping in touch with collaborative uavs:a deep reinforcement learning approach. 2018. *Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*.

118 X. Liu, Y. Liu e Y. Chen. Reinforcement learning in multiple-uav networks: Deployment and movement design. 2019. v. 68, n. 8, p. 8036–8049. *IEEE Transactions on Vehicular Technology, doi:10.1109/TVT.2019.2922849*.

119 M. Monajjemi, J. Bruce, S. A. Sadat, J. Wawerla e R. Vaughan. Uav, do you see me? establishing mutual attention between an uninstrumented human and an outdoor uav in flight. *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015. p. 3614–3620. *doi:10.1109/IROS.2015.7353882*.

120 N. Smolyanskiy, A. Kamenev, J. Smith e S. Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017. p. 4241–4247. *doi:10.1109/IROS.2017.8206285*.

121 N. I. Mowla, N. H. Tran, I. Doh e K. Chae. Afrl: Adaptive federated reinforcement learning for intelligent jamming defense in fanet. 2020. v. 22, n. 3, p. 244–258. *Journal of Communications and Networks, doi:10.1109/JCN.2020.000015*.

122 S. A. Hoseini, J. Hassan, A. Bokani e S. S. Kanhere. Trajectory optimization of flying energy sources using q-learning to recharge hotspot uavs. *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2020. p. 683–688. *doi:10.1109/INFOCOMWKSHPS50562.2020.9162834*.

123 Y. Cheng e Y. Song. Autonomous decision-making generation of uav based on soft actor-critic algorithm. *2020 39th Chinese Control Conference (CCC)*. 2020. p. 7350–7355. *doi:10.23919/CCC50068.2020.9188886*.

124 M. Freese S. James e A. J. Davison. Pyrep: Bringing v-rep to deep robot learning,. *arXiv preprint arXiv:1906.11176*. 2019.

125 A. Hernandez, C. Copot, R. De Keyser, T. Vlas e I. Nascu. Identification and path following control of an ar.drone quadrotor. *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)*. 2013. p. 583–588.

126 PYTORCH. *Pytorch.* 2021. Available from Internet: <https://pytorch.org/>.

127 S. L. Waslander D. Dostal J. S. Jang G. Hoffmann D. G. Rajnarayan e C. J. Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). 2004. v. 2, n. 12E, p. 4–121. *The 23rd Digital Avionics Systems Conference*.

128 Q. Lindsey N. Michael D. Mellinger e V. Kumar. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). 2010. v. 17, n. 10, p. 56–65. *Robotics Automation Magazine, IEEE*.

APPENDIX A – Publications

So far, we published one paper on the theme of quadrotors and machine learning:

1. *An Evolutionary Algorithm for Drone Trajectory Optimization in Aerial Challenges* - IEEE Latin American Robotics Symposium - SBR-LARS 2020