CENTRO DE INVESTIGACIONES EN OPTICA, A.C.

® POSGRADO INTERINSTITUCIONAL EN CIENCIA Y TECNOLOGÍA

DEEP Q-LEARNING IN ROBOTICS: A PATH PLANNING ALTERNATIVE

TESIS

QUE PARA OBTENER EL GRADO ACADÉMICO DE

MAESTRO EN CIENCIA Y TECNOLOGÍA EN LA ESPECIALIDAD DE MECATRÓNICA

PRESENTA
ING. GESEM ELIAB GUDIÑO MEJIA

DIRECTOR DE TESIS
DR. GERARDO RAMON FLORES COLUNGA

LEÓN, GUANAJUATO, AGOSTO, 2022.

POSGRADO INTERINSTITUCIONAL

Deep Q-Learning in Robotics: A path planning alternative

Acknowledgments

To Him who sits on the thrown, to him who lives forever and ever, be all the glory, be all the honor and be all the power.

Thanks to Consejo Nacional de Ciencia y Tecnología (CONACYT) who granted me the scholarship with which i was able to develop this work and continue with my professional formation. To all academic authorities of Centro de Investigaciones en Óptica (CIO). To Dr. Gerardo Flores, who was my advisor during the development ant thought me valuable lessons in the way. To my mother who always supported my dreams and ambitions, who pushed me forward even when i didn't wanted and gave up her dreams to live her sons. To my family, friends and colleagues that where beside me on this long and hard journey, thanks for your support.

Abstract

Literature shows that mobile robot research is an active field of study. An example is [1], which presents research on Human-Robot collaboration in the field of agriculture, and illustrates how this interaction can improve efficiency in agricultural activities. In addition, recent advances in the field of artificial intelligence (AI), have made it possible to find more efficient solutions in the field of robotics and computer vision. However, robotics is particularly challenging domain for any learning algorithm. A robot learning algorithm should be able converge quickly since it is difficult to carry out a million trials on such systems. It also should allow the robot to incrementally improve its performance while it is learning. In this work a development of such learning algorithms is presented. Using Deep Q-Learning, this work goes from simple robotic systems that learn how to balance a pendulum to more complex ones like performing path planning tasks that allow mobile robots to learn new behaviors in an initially unknown environment. Experiments on simulated environments have been presented and results have proved to be suitable for real world tests.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

The general hypothesis of this thesis is to investigate and proof that

Deep reinforcement learning algorithms can be used to train robotic systems to navigate autonomously in unknown environments and avoid obstacles that can be along their path, expanding the applications of autonomous robotic systems in the real world.

In order to do that, the following work is presented.

1.1 Overview and Motivation

Automation and mechanization are at a turning point thanks to the recent research growth in Artificial Intelligence (AI), this allows people to program tasks on computers to enable complex systems to learn how to perfect their skills through experience, just as we humans do.

In addition, several recent works on Human-AI and Human-Robot collaboration has motivated this work to investigate solutions for intelligent systems to interact with the world autonomously. In order to achieve that, very good path planning algorithms like the ones reviewed in this work have been developed.

1.2 Justification

Enabling robotic systems to interact autonomously in the real world brings enormous benefits, like speeding up manufacturing processes, making them more efficient and more cost-effective or reducing human danger exposure by developing aerial vehicles that can inspect high tension electric lines.

1.3 Goal of the Thesis

With this motivation in mind, the main goal of this work is as follows.

To investigate automatic learning methods capable of determining a collision free path that enables robotic systems to navigate on an unknown environment.

This can be subdivided into four objectives.

1. Exhaustively review the state of the art methods and theory behind automatic learning for robotics and path planning.
2. To find and study available methods, tools and technologies suitable for obstacle detection and path planning.
3. To implement the found algorithms in a simulation environment in order to extract results for evaluation.
4. Finally, to demonstrate the suitability for the application addressed in this thesis.

1.4 Problem Definition

Aiming to accomplish the task introduced in section 1.3, this thesis means to coverage the path planning problem in the particular for the case of an Unmanned Aerial Vehicle. This, in turn, arises the problem of avoiding obstacles that may be along the path in an unknown environment.

Therefore, in chapter 2 a review of previews studies on path planning for robotic systems is done. Then, in chapter 3, the methods along with tools and available technologies to overcome this problem are presented.

CHAPTER 2

Theory Fundamentals

The word ”Robot” was first introduced in a 1921 Czech science fiction play “Rossum’s Universal Robots” by Karel Capek [2]. Back then, robots where represented as androids while the Czech word was derived from the word for slave. But it was till 1954 when George C. Devol created and patented the first machine that we now consider as a robot [3]. The system was composed by a mechanical arm with a gripper, it was mounted on tracks and had a sequence of motions encoded by magnetic patterns [4].

Nowadays we understand the concept of a robot as a ”Mechanical system that can sense its environment and plan a set of actions in order to achieve a goal”. These actions can go from managing the speed of a quad-rotor’s motors to make it go to a desired position, to controlling the position of a robotic arm to make it reach a goal or grasp an object of interest.

In order to plan these sets of actions, the robot must have information about its environment. The way to achieve that, is through sensors that can obtain, and in some cases,

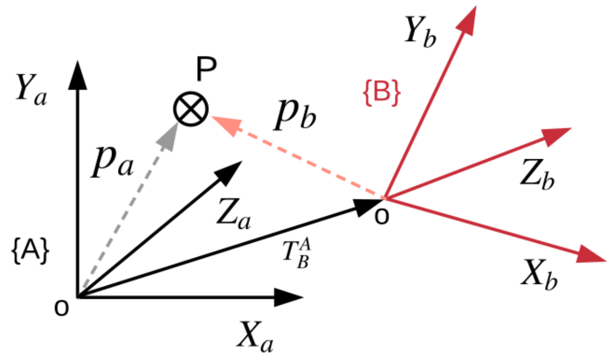


Figure 2.1: Relative position of point P w.r.t frames A or B.

even pre-process the data. Some examples of sensors can be a camera, an optical encoder or even a GPS.

After a sensor receives the information about the environment, its usually converted into an electrical signal that a controller board can read and interpret through a programmed algorithm. Then, another algorithm is in charge of taking the corresponding actions based on the robot state (where state refers to the final interpretation of all sensors readings).

2.1 Position and Orientation

A very important point in robotics is to represent objects position and orientation in space. For simplicity purposes an object can be represented as a point P in a coordinate frame A, just as it is shown in fig. 2.1.

However, if the same object is referenced under another coordinate system (point of view), e.g. B, it is necessary to find the homogeneous transformation of frame B w.r.t A as it can be seen in equation 2.1.

$$p_a = T_B^A \cdot p_b \tag{2.1}$$

Where p_a and p_b are the coordinate vectors of point P w.r.t frame A and B respectively, and T_B^A is the homogeneous transformation matrix of frame B w.r.t frame A, representing the rotation and translation of one frame with respect to the other. Equation 2.2 shows the general form of a homogeneous transformation matrix between frames A and B.

$$T_B^A = \begin{bmatrix} \mathrm{R}_{3\times 3} & \mathrm{t}_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.2}$$

Where vector t represents the translation of a body w.r.t a frame. In the same way, the rotation matrix R is used to describe the orientation of a body. This orientation is given in the form of a 3 dimensional vector containing the respective rotation angle for each of the rotation axis. This means that there exists three different rotation matrices, one for each axis [5].

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \quad R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \quad R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.2 Path planning concepts

A basic path planning goal is to generate a set of states that all together connect a start point or configuration S and a goal configuration G, while avoiding collisions with obstacles [6].

A configuration, in this context, describes the pose of a robot or robot state, while a configuration space C is the set of all the possible configurations. It's important to mention that if, for example, the robot and obstacle geometries are described in a 2D or 3D workspace, the path could be in a higher-dimensional configuration space. To understand more about this, look at the following examples.

If the robot can be modeled as a single point in a 2D plane or workspace, C will be as well a plane. Therefore C can be represented using two parameters (X,Y). Although, if the robot can be translated and rotated, even though the workspace is still in 2D, C is part of the special Euclidean group $SE(2) = R^2 \times SO(2)$, therefore, the configurations can be represented with parameters (x,y,θ). Another good example is, for instance, a robotic manipulator with n revolute joints. In this case C will be of n dimensions.

A set of configurations that is not in a collision state with obstacles is defined as a collision free space or C_f, then, $\overline{C_f}$ is the set of all forbidden configurations [7].

Then, a path planning algorithm should be able to find a configuration in C_f for the robot. In order to determine if a given configuration is in C_f, algorithms must be able to run collision detection tests to see if the robot's geometry is colliding with any part of the environment's geometry.

There are very well known algorithms that have been used in the context of this work such as Probabilistic Route Maps (PRM) [8] and Rapid Random Exploration Random Trees (RRT) [9]. Also, a large number of algorithms have been improved based on them. Like SRRT which resulted in a smoother trajectory, and faster exploration [10]. However in the case of RRT based methods, the randomness of exploration makes the trajectories planned by this algorithm have little consistency and sometimes be extremely time consuming.

For this reason, a lot of research has been done trying to find a consistent, fast and optimal method, such is the case of [11] where the authors present an algorithm called SD-SMAH* applied on a 6 degrees of freedom (DOF) robotic manipulator.

2.3 An Artificial Intelligence Alternative

Artificial Intelligence (AI) is the scientific field of computer science that focuses on the creation of programs and systems, which can exhibit behaviors considered intelligent or human-like. Typically, an AI system is capable of analyzing large amounts of data in order to identify patterns and trends. Therefore, it is possible to formulate predictions automatically, quickly and accurately.
Machine Learning (ML), according to Arthur L. Samuel in 1959, is defined as ”A field of study that gives computers the ability to learn without being explicitly programmed” [12].

There exists three basic types of machine learning:

- Supervised learning: This type of learning maps an input to an output based on given examples. The algorithm analyzes the training data iterately and produces a function which can be used for mapping new data.
- Reinforcement learning: Here the algorithm, called agent, takes actions that interact with a certain environment and produces a numerical value called reward. The general purpose is to find a function that maximizes the cumulative reward.
- Unsupervised learning: The model works on its own to find patterns in the given data. Unlike supervised learning it doesn’t require labeled data.

2.3.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are ML algorithms based on how biological neural networks work. Biological neurons receive signals through synapses located on the membrane of the neuron. When the signal received is strong enough, the neuron activates and emits a signal, this signal might be sent to another synapse, and activate other neurons.

When speaking of artificial neurons, the above concepts are applied in a basic way. Then, artificial neurons consist basically of inputs, which then are multiplied by weights to simulate the strength of the signal, and then computed by a mathematical function that determines if the neuron activates or not. With this, it can be deduced that by adjusting the weight of an artificial neuron, a desired output for a specific input can be obtained. The mathematical operations that these neurons commonly compute are described as follows $\phi(\sum_i^n w_i x_i)$ where w_i is the weight of neuron i, x_i is the value of neuron i and ϕ is a non-linear activation function [13].

An ANN then consists of individual artificial neurons located in a series of clusters called layers that, in some way, are connected to each other. A common architecture is when the neurons in each layer are connected to neurons in the next layer. Each individual neuron performs the mathematical computation described above and then transmits its results to all the nodes to which it is connected [14]. A diagram that shows a common fully connected ANN is shown in figure 2.2.

This ANN's typically propagate their signals to the next layer as mentioned above, this action is known as feeding forward the network. To find the correct set of weights for an ANN, an algorithm called back-propagation is commonly used. In simple terms, after each feed forward, the back-propagation method optimizes the network parameters (weights and biases) using gradient decent over the total error of the network [15]. The total error of the network can be obtained using the mean square error function.

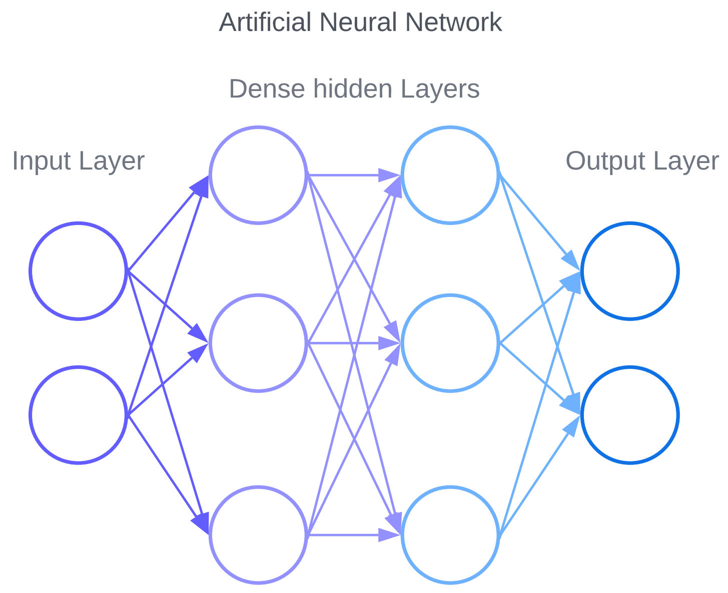


Figure 2.2: Diagram showing a simple fully connected ANN.

$$E = \frac{1}{2n}\sum_{i}^{n}(y_i - \widehat{y}_i)^2 \tag{2.3}$$

Where y_i is the desired output, $\hat{y}_i$ is the ANN output and n is the total number of output neurons. Finally, to update the weights of the network, back-propagation method uses gradient decent over the error with respect to the weights as follows $\theta^{t+1} = \theta^t - \alpha\frac{\partial E}{\partial \theta}$ where θ is the set of all weight in the network, θ^t are the weights in step t and α is a learning rate factor.

2.3.2 Reinforcement Learning

When speaking about Reinforcement Learning (RL), the first thing that may pop in to our minds is an interactive learning, where there is an instructor observing the behavior of a trainee, and depending on this observation the instructor offers a positive or negative reward, all this while the trainee tries to follow a set of instructions. Well, this is not far from the actual definition of RL. “RL learns how to map situations to actions so as to maximize a numerical reward signal" [16].

Now, lets define the trainee as an agent, as it is usually called in RL. The ultimate goal of the agent is to maximize its reward over time. With a correct reward function, the agent can achieve a good performance on the task that has been given. This function depends on the observations that come from the interaction of the agent with its environment. So, the agent on a state s_t takes an action a that leads it to sate s_{t+1} (interaction with environment), then given this transition the reward function r computes the corresponding reward, and so on. This cycle can be observed in fig 2.3.

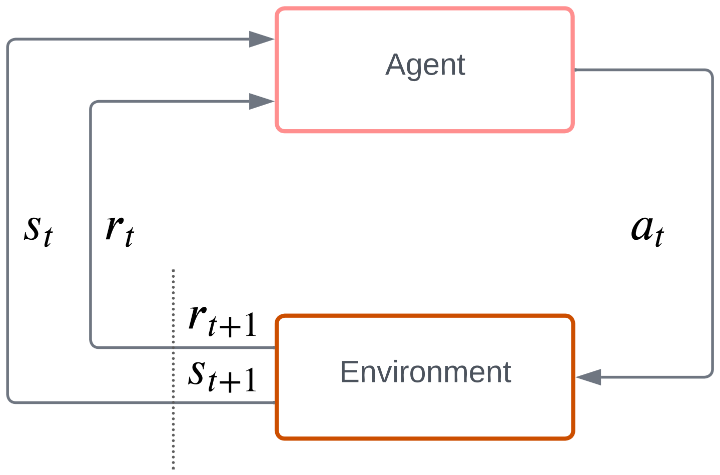


Figure 2.3: Figure showing a RL learning cycle, where the agent is the algorithm that makes the decision of what action to take in an instant of time given a state and a reward. The environment is everything outside the agent.

RL can broadly be separated into two main groups: model free and model based algorithms. The main difference between these two algorithms is that model free do not learn transition functions to make predictions of future states and rewards. Good examples of a model free algorithms are Q-Learning and Deep Q-Learning.

2.3.3 Deep Q-Learning

In Q-Learning one of the main goals is to learn the set of Q-Values for an environment. The Q-value is determined by bellman's equation as follows [17].

$$Q(s,a) = r(s,a) + \gamma \max_a Q_{t+1}(s_{t+1}, a_{t+1}) \tag{2.4}$$

Where γ is a discount factor which controls the contribution of reward r further in the future, s and a are current state and action respectively as mentioned in the previous section. In other words, the Q-value given by $Q(s,a)$ is the immediate reward r plus the highest Q-value possible from the next state s_{t+1}, and serves as an indication for how good it is for an agent to pick a while being in s. Explicitly choosing the best known action at s is known as exploitation.

One of the challenges that arise in RL, is the trade-off between exploration and exploitation. The dilemma is that neither of them can be pursued exclusively, exploitation maximizes the expected reward on the immediate term, but exploration may produce a better reward accumulation in the long run.

A common strategy for tackling this problem is the Epsilon Greedy Exploration Strategy. Given a probability ε, the agent selects a random action (explore), and, with probability $1-\varepsilon$, it selects the action that has a maximum Q-value (exploit).

In Deep Q-learning, a neural network maps input states to action/Q-value pairs. However, it is important to note that since in Deep Q-Learning the main objective of the ANN is to get good action/Q-value pairs, the target $\widehat{y_i}$ in the loss function E presented in equation 2.3 will be $Q(s,a|\theta)$ where θ still represent the network weights. Then, the complete loss function will be as follows [18].

$$E = (r(s,a) + \gamma \max_a Q(s_{t+1}, a_{t+1}|\theta) - Q(s,a|\theta))^2 \tag{2.5}$$

Where the left hand side of the subtraction is the target Q-value and the right hand side is the predicted Q-value. Since the target is not static, because RL doesn't use pre-labeled data, and actually, it is frequently changing as the agent continuously learns the Q-value and action ground truths. The training is not stable. To overcome this issue, a separate network is used to estimate the target. This target network has the same architecture but with frozen weights. Then, every n steps, the weights from the main prediction network are copied to the target network. This leads to more stable training, keeping the target function fixed for n steps. Please see figure 2.4 for a better understanding of this process.

Next, to update the prediction and target networks, Deep Q-Learning agents use a technique called Experience Replay. This means that instead of running the algorithm on each state/action pair as they occur, the agent stores experiences with the form $[s, a, r, s_{t+1}]$ in a replay buffer and then trains over a small batch that is randomly sampled from it.

Summarizing the Deep Q-Learning process into steps, it will be as follows.

1. Feed the state s to the Deep Q Network.
2. Select an action a using the epsilon-greedy policy, the system will interact with its environment and return a reward r given the reward function as shown in figure 2.3.
3. Store this transition in with the form $[s, a, r, s_{t+1}]$ in a replay buffer.
4. Sample random batch from the replay buffer and compute E using equation 2.5.
5. Update the prediction network weights using back-propagation and gradient decent algorithm to reduce E.

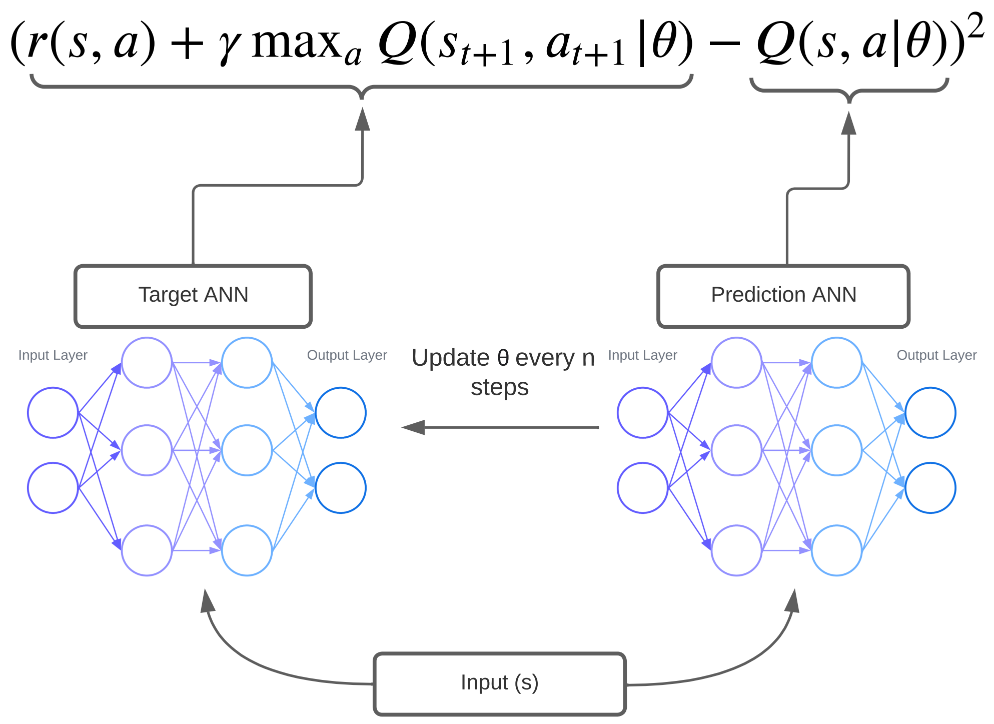


Figure 2.4: Diagram showing the objectives for the target and prediction networks in a Deep Q-Learning training.

6. Copy the prediction network weights to the target network every n steps

7. Replay until getting the maximum accumulated reward

2.3.4 Applications

RL alongside ANN in robotics has been growing enormously in the past years. Applications go from robotic arm domain, like learning how to grasp objects in cluttered environments [19] or perform high precision assembly tasks [20], to autonomous flight domain, where RL is being used to design controllers for hybrid air crafts like in [21] or [22] where RL is used for autonomous landing on moving platforms.

Of course, path planning is not the exception. Enabling robots to autonomously navigate in different environments is essential for real-world robotic deployments. Traditional navigation frameworks employ Simultaneous Localization and Mapping (SLAM) [23] to map an unknown environment to then move to their destination using path planning algorithms. However, recent works that use Deep RL have shown to be promising for this field, like in [24] [25] [26] where a mobile robot learns how to move around an unknown environment using Deep RL.

As it was mentioned before, an important part of path planning algorithms is obstacle avoidance. A very interesting work on this topic has been presented in [27], where an obstacle avoidance algorithm based on monocular vision and trained with Deep RL is developed. However, the use of an image as input for an algorithm often requires bigger computational power than other sensors. That's why in [28] a path planning algorithm using 2D-laser data as input and Deep RL has demonstrated to work on both simulated and real world environments.

CHAPTER 3

Methods and Results

In this section, the experimental process implemented to achieve the goals stated in section 1.3 is described.

As mentioned in the previous chapter, reinforcement learning is a technique based on the interaction of the system (in this context: Robotic) with an environment. Consequently, one of the challenges to overcome is the lack of control over the robot's movements during training, which, due to cost and safety issues, has led several research groups around the world to develop simulation platforms and tools, with which training can be carried out safely and at lower cost.

Therefore, in this work, a set of open source software tools for robotic simulations and RL trainings are used. These tools are explained below.

The Open AI developers team has created a set of tools called Gym [29], which has gained popularity among the community of ML algorithm developers. These open source tools allow the user to use virtual environments to implement their RL algorithms, as well as compare results with those of a large community of developers around the world.

On the other hand, robotic systems are composed of a large number of sensors and actuators with different communication protocols, frequencies and data types. That is why, in this work, it was decided to use the Robotic Operating System (ROS) [30]. This open source framework provides simulation and data visualization tools, hardware abstraction functionality, controllers, and communication between modular processes called nodes through TCP protocol.

Communication in ROS is based on a client-server architecture within a local area network (LAN), so that different nodes can send structured information through messages in topics, and, as well, some other process can obtain such data by registering as a subscriber to that same topic. Thanks to ROS and its large and active community, complex software architectures can be built while maintaining the advantages of modularity and open source software.

In [31] a breakthrough in the topic of simulation and learning has been presented, where tools such as ROS and simulation platforms such as Gazebo are used to extend Gym capabilities. However, when using virtual environments for training, new challenges are generated. Sim to real, is a research area that studies how to approximate the behavior of dynamic systems in a simulation to real systems. In [32] and [33], techniques that get very close approximations to the behaviors of a real system are presented.

Therefore, this work starts with simple systems training on simulation. As a first experiment, the training of an inverted pendulum system is presented.

3.1 Experiment A. Inverted Pendulum System

The inverted pendulum system is a well known mechanism that consists of a rail on which a cart can slide, and, at the same time, this cart carries a pendulum that can rotate freely. The system must be instrumented so that the angle of the pendulum with respect to the normal can be obtained, as well as the position and velocity of the cart. A diagram of such a system can be seen in figure 3.1.

The goal of this training then, is to keep the pendulum vertical, or, in other words, balanced, by controlling its displacement along the rail. Having said that, the observation space for this system will be continuous and consists of the position of the cart along the rail, its linear velocity and pendulums angle position and angular velocity. As for the action space, there are two discrete actions, which correspond to a constant velocity in each direction (X,Y). Since this is a reinforcement learning simulation, each of the observations has a range within which the agent can consider that the episode has not ended. These ranges can be observed in the table 3.1.

Observation	Min	Max
Cart position (x)	-4.8	4.8
Cart velocity (v)	-∞	∞
Pendulum angle (θ)	-24 grad	24 grad
Pendulum angle velocity (ω)	-∞	∞
Action	**Discrete value**	
Left movement	0	
Right movement	1	

Table 3.1: Observation and action spaces for an inverted pendulum system.

For this system's training, the deep reinforcement learning method known as DeepQ-learning was used with the following training parameters. $\gamma = 0.95$, $\alpha = 0.001$, $\varepsilon_{max} = 1.0$, $\varepsilon_{min} = 0.01$, $\varepsilon_{decay} = 0.95$. Where γ represents the discount factor, α represents the learning rate, and the values of ε represent the maximum and minimum values of the exploration

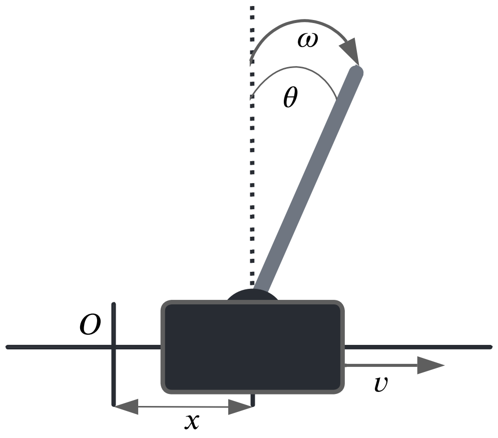


Figure 3.1: Diagram showing the variables for an inverted pendulum system.

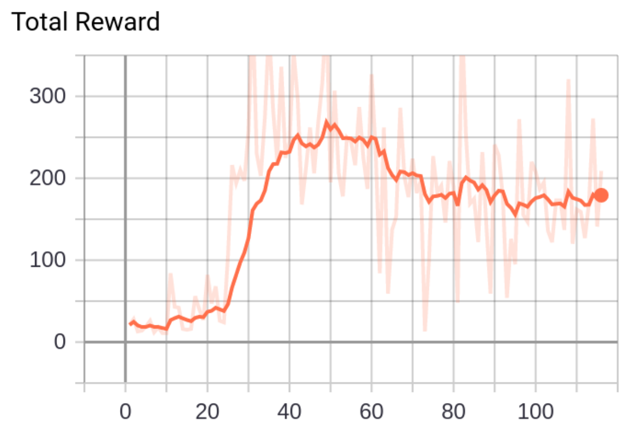
factor and its decay factor over training respectively.

Using these values and through a training of about 100 episodes, good results were obtained, accumulating an average reward of 195, which means that that the pendulum remained balanced for an average of 195 steps in each episode. This, given that the reward function for this experiment was equation 3.1.

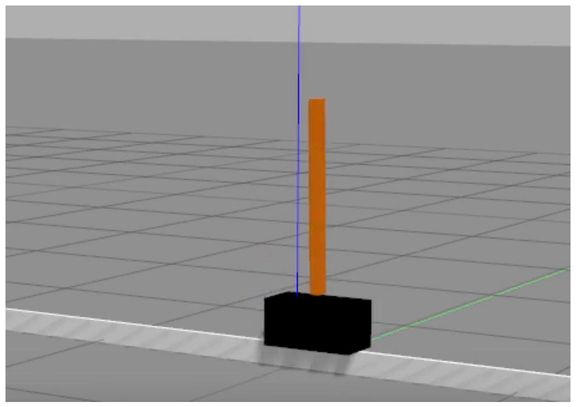
$$R = \begin{cases} 1 & \text{if } -24 < \theta < 24 \\ 0 & \text{else} \end{cases} \tag{3.1}$$

The result of this training in terms of the accumulated reward per episode can be seen in figure 3.2a. Likewise, a snapshot of the model in Gazebo simulation environment can be seen in figure 3.2b.

After obtaining these results in a relatively simple system, with respect to the complexity of the observation and action spaces and the overall goal, it was decided to continue with the experiments in a more complex system.



(a) Training chart showing the accumulated average reward per episode.



(b) Figure showing the cart balancing the pendulum in Gazebos simulation environment.

Figure 3.2: Results obtained for an inverted pendulum system after a training of approximately 100 episodes.

3.2 Experiment B. Cubli Robotic System

For the following training, a system known as cubli [34] was used. This system consists of a cube with 3 rotating disks inside, each of which is designed to control one of the different euler angles, roll angle ϕ, pitch angle θ, and yaw angle ψ. This small robotic system works by accelerating each of the momentum disks to then stop and transfer the angular momentum generated to the cubes frame, causing it to rotate in the direction and orientation of the disk.

In order to keep the experiment simple, only one momentum disk will be used, the one corresponding to ϕ. Thus, the objective of this training will be to control the rotations direction of the disk, its velocity and whether it brakes or not, this in order to make the cube advance the greatest amount of distance in the direction corresponding to ϕ represented as Y. In the same way as in the previous experiment, the system will have constraints that will define whether an episode is finished or not, these constraints are given by the allowed ranges for each observation within the observation space. The observations and actions spaces for this experiment are described in table 3.2.

Observations	Min	Max
Disk velocity	$-\infty$	∞
Distance in Y	0	2.0 m
Lineal Velocity in Y	$-\infty$	∞
ϕ	0	2π rad
θ	0	0.2 rad
ψ	0	0.1 rad
Action	**Discrete value**	
Rotate the disk cw	0	
Rotate the disk ccw	1	
Brake disk	2	
Increase velocity	3	
Decrease velocity	4	

Table 3.2: Observation and action spaces for Cubli robotic system.

Once the observation and action spaces were defined, the reward function was defined as follows

$$R = \Delta_Y + V_Y - |\sin\psi| \tag{3.2}$$

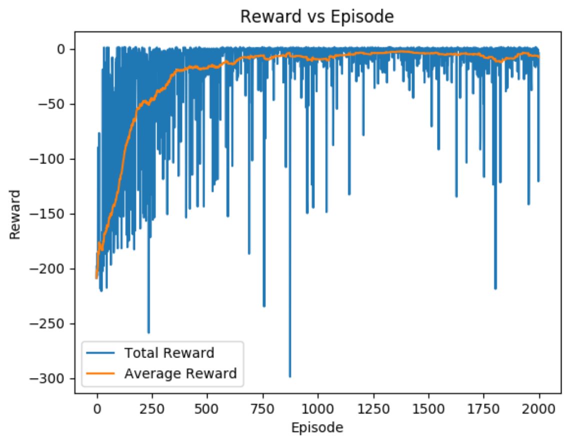
where Δ_Y represents the displacement along Y and V_Y is linear velocity in Y.

Next, several training sessions were carried out with the goal of observing and learning more about the interaction of robotic systems with their environment and how this is related to reinforcement learning. Being a more complex robotic system due to the number of observation variables and number of possible actions, the results with respect to the cumulative reward were less satisfactory than in the previous experiment.

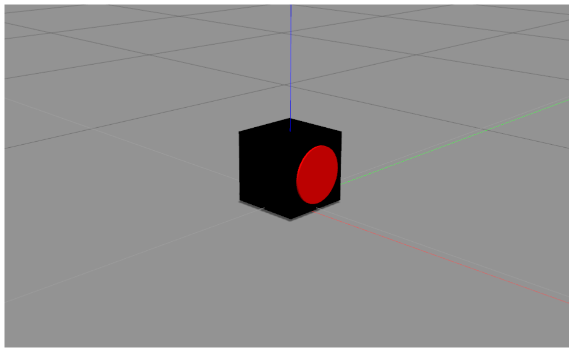
In this experiment it was observed that although a slight improvement in reward accumulation was noticed, the cubli was not able to reach the desired value. However, the system did increase the traveled distance on Y-axis as training progressed. This behavior was because the agent found a sub-optimal solution to the problem. Causing the cubli system to vibrate in the Y direction to achieve the goal.

Given this result, the reward function was modified by adding w_1, w_2, w_3 to appropriately weight the different observations and influence cublis behavior during the training. The new reward function is as follows $R = \Delta_Y w_1 + V_Y 1_2 - |\sin\psi| w_3$. Where values of $2.0, 1.0$ and 0.5 were assigned for the weights w_1, w_2, w_3 respectively.

After training the model with this new reward function, better results were obtained, making the cubli system advanced significantly in the Y direction. Likewise, at the end of the training, the cubli system was able to make some complete turns in ϕ, achieving the training goal. Please refer to figure 3.3 to see the final training results and a screen capture of the environment in which the training was performed.



(a) Training chart showing the accumulated average reward per episode.



(b) Figure showing the Cubli robotic system in Gazebo simulation environment.The Black cube represents Cublis frame and the red disk represents the momentum disk corresponding to ϕ.

Figure 3.3: Results obtained for Cubli robotic system after a training of approximately 2000 episodes.

Through this experiment, the importance of the reward function and the impact it has on the behavior of the robotic system during training was demonstrated.

3.3 Experiment C. 3 DOF Robotic Arm

The goal of this experiment is to make a 3 degrees of freedom (DOF) robotic arm learn a policy that allows it to position its end effector (e) as close as possible to the goal (g), in other words, to reduce the distance between both points (d), this through the configuration of its joints, where each joint has an angular position called $\theta_1, \theta_2, \theta_3$ respectively.

Figure 3.4b shows a snapshot of the virtual environment used to carry out the training, where the green dot represents the position e and the red dot the position g. It should be noted that at the end of each episode the robot must return to an initial position given by a fixed configuration, called $(\theta_{1_i}, \theta_{2_i}, \theta_{3_i})$. Likewise g varies randomly within the configuration space of the robot.

For the development of this experiment, the observations and action spaces were designed. In this case, discrete actions were chose to reduce the complexity of the algorithm to be used. Then, the actions space is composed of 8 different actions, which correspond to the increment or decrement of each joint plus the increment or decrement of all the joints at the same time. This increment is defined as δ. In the other hand, the position of the goal, the position of the end effector and the Euclidean distance between them were taken as observations. A list of the actions and observations spaces can be seen in the table 3.3.

Once the observations and actions spaces were defined, the ANN to be used for the training of the robotic arm was designed. Since the observations space is very small, it was chosen to design a very simple architecture with few layers to avoid gradient vanishing problems and reduce the amount of processing required between each action. The ANN architecture can be seen in the table 3.4.

Recalling that the main goal of the RL is to maximize the long-term reward, the reward

Observations	Min - Max
Goal position (g_x, g_y, g_z)	$-\infty/\infty$
End effector position (e_x, e_y, e_z)	$-\infty/\infty$
Distance between end effector and goal (d)	$-\infty/\infty$
Action	**Discrete value**
Increase joint 1	0
Decrease joint 1	1
Increase joint 2	2
Decrease joint 2	3
Increase joint 3	4
Decrease joint 3	5
Increase all joints	6
Decrease all joints	7

Table 3.3: Observation and action spaces fro a 3 DOF robotic arm.

Layer	Neurons	Activation function
Input layer	6	ReLU
Dense layer	64	ReLU
Dense layer	64	ReLU
Dense layer	32	ReLU
Output layer	8	SoftMax

Table 3.4: Table showing the ANN architecture fro this experiment.

function is a very important part, as it defines what the agent will consider as good or bad decisions. Then, given the goal of this experiment, the following reward function was defined.

$$R = \begin{cases} -1 & \text{if} \quad d_t \geq d_{t-1} \\ 0 & \text{if} \quad d_t < d_{t-1} \\ 1 & \text{if} \ -\sigma < d_t < \sigma \end{cases} \tag{3.3}$$

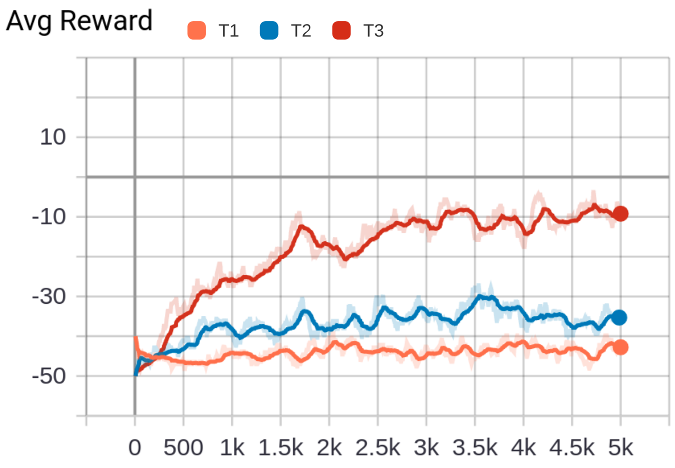
Where σ represents the tolerance value at which the end-effector is considered to have reached the goal position. d_t and d_{t-1} are the distances from the end-effector to the target point at time t and $t-1$ respectively and R is the reward value.

In order to find the correct learning parameters, several trainings were conducted with different configurations. A table with the 3 best configurations is presented in table 3.5.

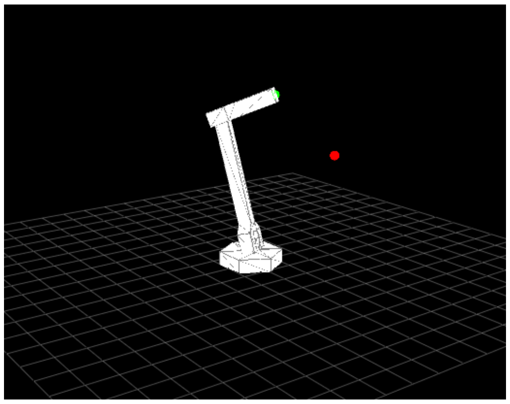
Training	α	γ	ε_{max}	ε_{min}	ε_{decay}	δ
T1	0.001	0.2	0.7	0.01	0.9995	0.01
T2	0.001	0.9	0.5	0.01	0.9995	0.01
T3	0.001	0.95	0.5	0.01	0.9995	0.05

Table 3.5: Table showing 3 different configurations for training.

Each training was 5 thousand episodes. In the end, a satisfactory result was achieved, the end effector of the robotic arm was able to reach σ tolerance range for the goal position in each episode. A comparison of the results obtained in terms of the cumulative reward of the three trainings can be seen in figure 3.4a.



(a) A comparison chart between T1, T2and T3 training configuration in terms of average cumulative reward per episode.



(b) Figure showing the virtual environment used for the training of a 3 DOF robotic arm

Figure 3.4: Results obtained for a 3 DOF robotic arm after a training of 5 thousand episodes.

It can be observed that the best training was T3, bringing its final average value close to 0, demonstrating that most of the steps within the episodes resulted in a decrease of d according to the equation 3.3, achieving the objective of positioning e within the σ range.

Next, the last experiment of the work is presented, being a proposal for the solution of the path planning problem for Unmanned Aerial Vehicles a.k.a UAVs.

3.4 Experiment D. Path Planning for a UAV

The main goal of this experiment is to find an obstacle-free path that a UAV can follow to avoid collisions. The available inputs or observations are the position of the UAV in space, and the position of these obstacles with respect to the UAV. In a real case, the position of these obstacles will be given by sensors that the UAV carries with it, these can be RGB-D type sensors, stereoscopic cameras, or LiDAR type radars among others. These sensors are limited by a detection range, this means that the environment in which our agent can act, taking into account that the positions of the obstacles are required as an observation, is limited by this range that will be defined as λ.

Given this constraint and for the sake of simplicity, it was decided to model the environment as a three-dimensional grid, where the size is defined by λ. This in turn allows us to define the observations and actions spaces as discrete, and further simplifying the problem. Each cell in this space is separated by a constant distance denoted as δ (not to be confused with the δ mentioned in the previous experiment). Together, this generates a network of nodes that each represent a position in space to which the UAV can move. A simplified version of the above environment can be seen in figure 3.5.

Then, the agent starts at a random position at one end of the cube and is assigned a target position, with this, the agent must find a sequence of actions (explained below) that at the

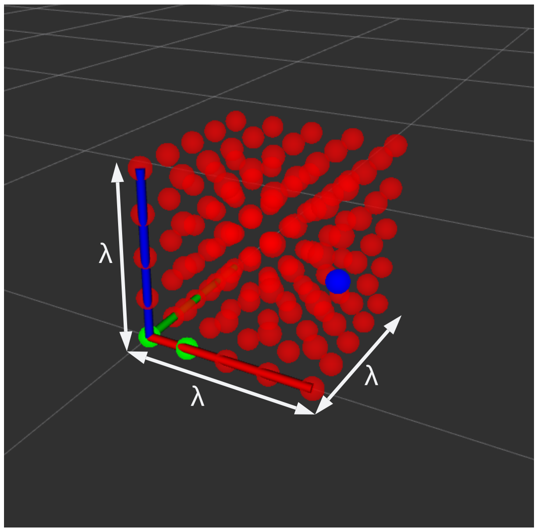


Figure 3.5: Image showing a simplified version of the environment used in this experiment. Each red dot represents a position within the UAV workspace defined by λ, the blue dot represents the target coordinate and the frame shows the origin point from where the agent should start the planning.

end can constitute a path from the starting point to the target point.

To achieve this goal, this experiment was divided into 4 parts, each one more complex than the previous one, starting with a small workspace without obstacles, until reaching a size close to a real world environment, with obstacles with positions that vary in each episode. The table below shows the main differences between these experiments.

Experiment	λ	Obstacles	δ
D.1	0.5	0	0.125
D.2	2.0	0	0.1
D.3	3.0	1	0.1
D.4	3.0	2	0.1 y 0.5

Table 3.6: Comparison table between 4 different sub-experiments

3.4.1 Experiment D.1

This experiment is the simplest, having $\lambda = 0.5m$, a $\delta = 0.125m$ and no obstacles. Once the objective and the strategy to be used were defined, the observations and actions spaces were designed. The action space then consists of 6 different actions corresponding to the basic displacements (forward, backward, up, down, right, left). On the other hand, the observations space is composed of the agent's current position and the target position. It is important to emphasize that both are always inside the workspace, represented as the red cube in the figure 3.5. The observation and action spaces are explained in table 3.7.

Observation	Min	Max
a_x	0	λ
a_y	0	λ
a_z	0	λ
g_x	λ	λ
g_y	0	λ
g_z	0	λ
Action	**Discrete value**	
Up	0	
Down	1	
Left	2	
Right	3	
Forward	4	
Backward	5	

Table 3.7: Observations and actions spaces, where (a_x, a_y, a_z) represents the agents position a, and (g_x, g_y, g_z) the goal or targets position g.

For this experiment, the reward function gives a value of -1 for each step in which the agent has not reached the target position, in addition, a value of $-\frac{d}{10}$ is added where d is the Euclidean distance from the agent's current position to the target position. This is done in order to negatively weight the time in which the agent achieves its task and, as well, negatively weight long distances within the path. Thus, the agent is encouraged to reach the target in the less amount of steps and traveling the shortest possible distance.

On the other hand, a value of 10 is awarded if the agent manages to reach the target. The reward function is described by the following equation.

$$R = \begin{cases} -1 - \frac{d_t}{10} & \text{if } d_t > \sigma \\ 10 & \text{if } d_t \leq \sigma \end{cases} \tag{3.4}$$

Next, and for the same reasons as the previous experiments, it was decided to design a small neural network with few layers. The architecture can be seen in the table below.

Layer	Neurons	Activation function
Input layer	6	ReLU
Desne layer	64	ReLU
Desne layer	64	ReLU
Output layer	6	SoftMax

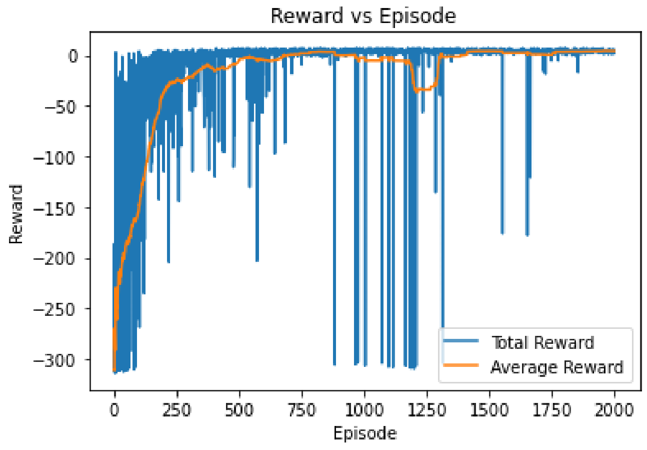
Table 3.8: ANN architecture for a path planning task training.

In addition to the design of the neural network to be used, it is important to define the learning parameters for training. The list of parameters used for this experiment can be found in table 3.9.

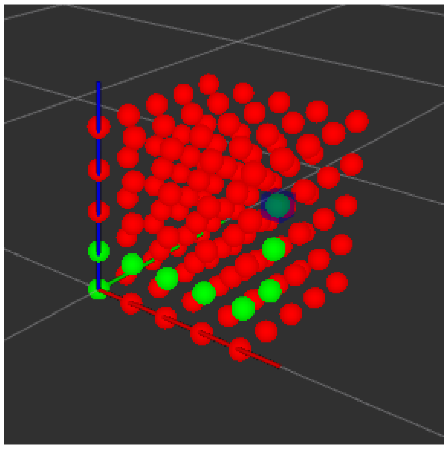
α	γ	ε_{max}	ε_{min}	ε_{decay}
0.001	0.99	1.0	0.01	0.9995

Table 3.9: Table showing the training parameters used for the training of D.1.

Then, the model was trained with the simplified environment shown in figure 3.5. This training required 2000 episodes with a maximum of 100 steps per episode in about 2min 34sec. The average cumulative reward value at the end of the training was 3.94. These values can be seen in figure 3.6a and 3.6b.



(a) Figure showing the accumulated reward per episode during the agent's learning process over 2,000 episodes.



(b) Figure showing the path obtained by the agent after training, where the set of green nodes represents the path from the initial point to the target.

Figure 3.6: Figure showing results for a workspace with $lambda = 0.5m$ and $delta = 0.125m$.

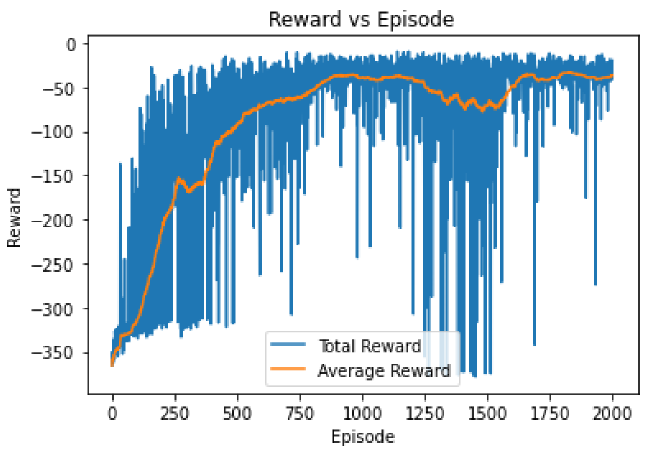
After obtaining very good results in a relatively short time, it was decided to continue with the next part of the experiment.

3.4.2 Experiment D.2

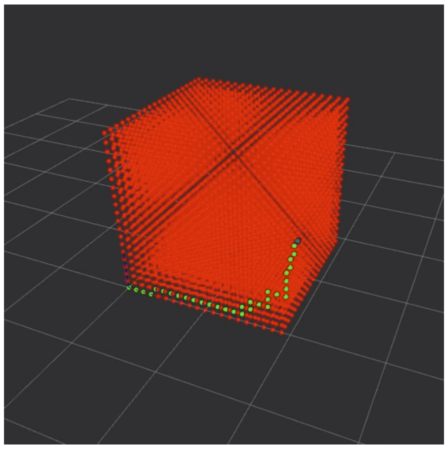
For experiment D.2, the training parameters shown in table 3.9 as well as the neural network shown in table 3.8 were used. This training required 3000 episodes with a maximum of 300 steps per episode in about 31min 20sec. Its average reward value per episode was -36.72. The results of the training can be seen in figs. 3.7a and 3.7b.

3.4.3 Experiment D.3

For D.3 the workspace was expanded to $\lambda = 3m$ and $\delta = 0.1m$, and an obstacle O of $1.5m^3$ was included. It is worth noting that for this experiment the positions of the obstacle and



(a) Figure showing the accumulated reward per episode during the agent's learning process.



(b) Figure showing the final path obtained after training, where the set of green nodes represent the trajectory from the initial point to the target.

Figure 3.7: Figure showing results for a workspace with $\lambda = 2m$, and $\delta = 0.1m$.

the target vary in each episode, in order to achieve a general policy.

By adding these new variables, it is necessary to update the observations space. So, for D.3 the observation space is defined in the table 3.10. Being a cube-shaped obstacle, the position of O must be represented by the two corners that mark its starting and end points, these corners are denoted as O_i and O_f respectively.

In the same way, when adding an obstacle to the environment, it is necessary to define a collision in the following form $O_i < a < O_f$. It is important to mention that in order to be a real collision, all the components of the vector a must fulfill the inequality at the same time. In this case, since the agent is sought to find a collision-free policy, the actions will be restricted to the collision-free space. This is achieved as follows.

When the selected action brings the agent into a collision state, the agent retains its current position, simulating a crash effect, or in other words, it doesn't decrease d. Thus the

Observation	Min	Max
a_x	0	λ
a_y	0	λ
a_z	0	λ
g_x	λ	λ
g_y	0	λ
g_z	0	λ
O_{i_x}	0	$\lambda - 1.5$
O_{i_y}	0	$\lambda - 1.5$
O_{i_z}	0	$\lambda - 1.5$
O_{f_x}	1.5	λ
O_{f_y}	1.5	λ
O_{f_z}	1.5	λ

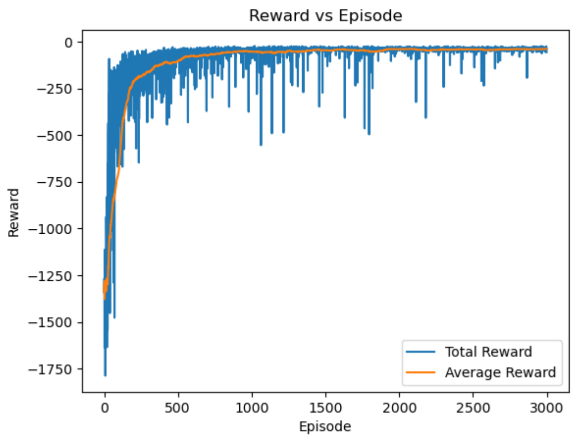
Table 3.10: Observation sapce, where (a_x, a_y, a_z) represent the position of the agent a, (g_x, g_y, g_z) the target position g, and the coordinates $(O_{i_x}, O_{i_y}, O_{i_z})$ and $(O_{f_x}, O_{f_y}, O_{f_z})$ the two corners defining the obstacle O.

agent will receive a negative reward given the equation 3.4. This equation motivates the agent to evade actions that lead to any state that does not reduce d and thus, in this case, to evade the obstacle.

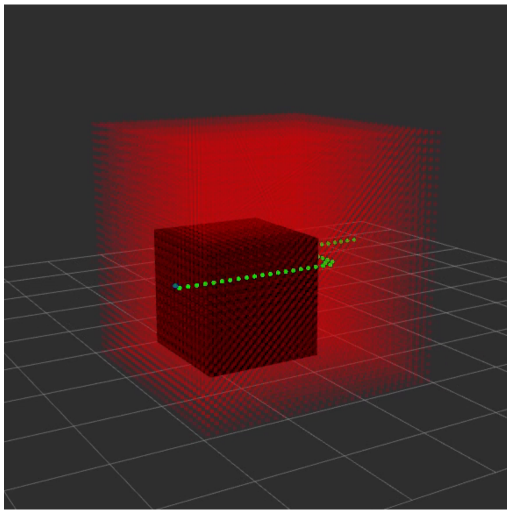
Following these rules and with the same training parameters and neural network as the previous experiment, good results were obtained and can be seen in figure 3.8, where the average cumulative reward value at the end of training was -55.76. To obtain these results, $3,000$ episodes with a maximum of 300 steps per episode in about 2hrs 37min were needed.

3.4.4 Experiment D.4

,For this experiment and motivated by the work presented in [35], two cylinder-shaped obstacles with a height of λ and a radius of Θ are included. The design of these obstacles has two main objectives, 1. to represent the environment in a closer way to reality, trying to emulate power line poles, trees or any static object that can be represented as a cylinder, and 2. this design allows to significantly reduce the observations space than if there where



(a) Figure showing the accumulated reward per episode during the agent's learning process.



(b) Figure showing the final path obtained after training, where the set of green nodes represent the path from the initial point to the target.

Figure 3.8: Figure showing results for a workspace with $\lambda = 3m$, $\delta = 0.1m$ and an obstacle O of $1.5m^3$ represented as a black cube within the workspace.

two cube or any other shape obstacles and therefore simplify the learning process.

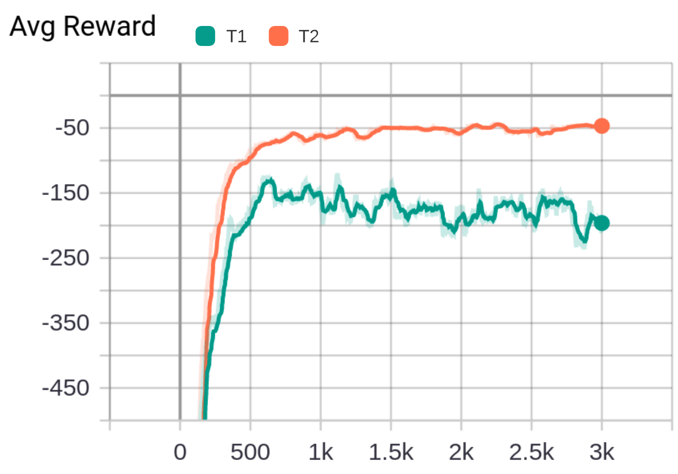
Then, observations space for this experiment, unlike the previous one, contains the positions in X and Y of the center of each cylinder and the distance from the position of the agent to each obstacle edge, this distance is given by $d_O = \sqrt{(a_x - O_x)^2 + (a_y - O_y)^2} - \Theta$ for each obstacle. The observations space for D.4 can be seen in the table 3.11.

Once the observations space for this experiment was defined, the training was executed in the same way as the previous one. In order to achieve good results, several training with different parameters were carried out. The two configurations that gave the best results are shown in table 3.12.

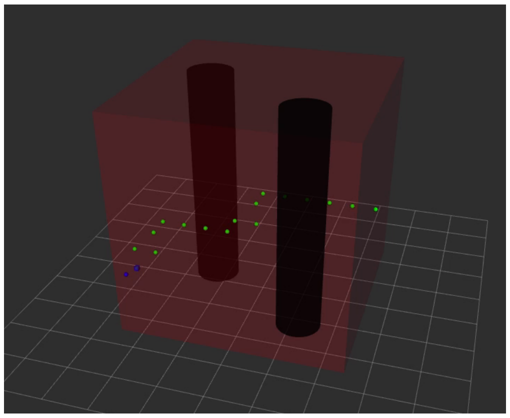
After T1, learning was improved by significantly reducing the dimension of the action space by increasing δ by five times as can be seen in table 3.12. This effect resulted in better

Observation	Min	Max
a_x	0	λ
a_y	0	λ
a_z	0	λ
g_x	λ	λ
g_y	0	λ
g_z	0	λ
O_{1_x}	Θ	$\lambda - \Theta$
O_{1_y}	Θ	$\lambda - \Theta$
d_{O_1}	Θ	$\sqrt{(\lambda-\Theta)^2+(\lambda-\Theta)^2}$
O_{2_x}	Θ	$\lambda - \Theta$
O_{2_y}	Θ	$\lambda - \Theta$
d_{O_2}	Θ	$\sqrt{(\lambda-\Theta)^2+(\lambda-\Theta)^2}$

Table 3.11: Observations space, where (a_x, a_y, a_z) represents the agents a position, (g_x, g_y, g_z) the targets g position, coordinates (O_{1_x}, O_{1_y}) and (O_{2_x}, O_{2_y}) the centers of each obstacle respectively and d_{O_1} and d_{O_2} the Euclidean distances from the agent to the edge of each obstacle.



(a) Figure comparing T1 and T2 trainings in terms of the average cumulative reward per episode. Differences between T1 and T2 can be seen in the table 3.12.



(b) Final path obtained after training T2, where the set of green nodes represent the path from the initial point to the target and the black cylinders within the workspace represent obstacles O_1 and O_2.

Figure 3.9: Figure showing the results obtained after a $3,000$ episodes training for a path planing algorithm with two obstacles in its workspace.

training results as seen in figure 3.9a, as well as a reduction in training time by approximately 3 times lasting a total of 7min 14sec. On the other hand, this modification will not

Training	α	γ	ε_{max}	ε_{min}	ε_{decay}	δ
T1	0.001	0.99	1.0	0.01	0.9995	0.1
T2	0.002	0.95	0.5	0.01	0.9975	0.5

Table 3.12: Table showing differences between trainings T1 and T2 for this experiment

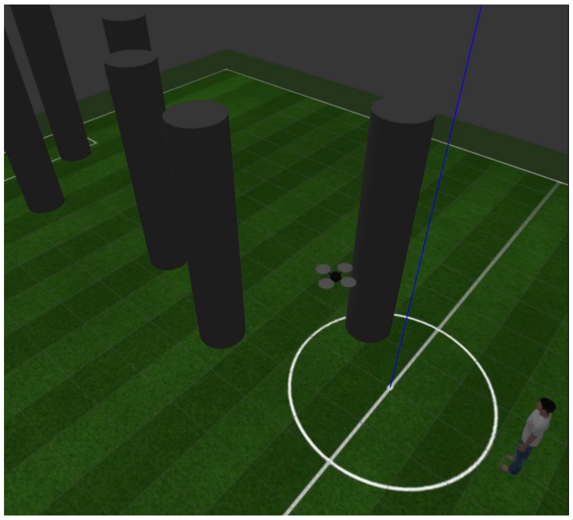
affect the overall goal of the experiment, since in a real scenario the UAV's receive waypoints much farther apart.

In order to test the reliability of the algorithm, the policy (in this case the trained ANN) was run in 3 trials with 1000 random start and target points each. The results for T1 and T2 are listed in the table below.

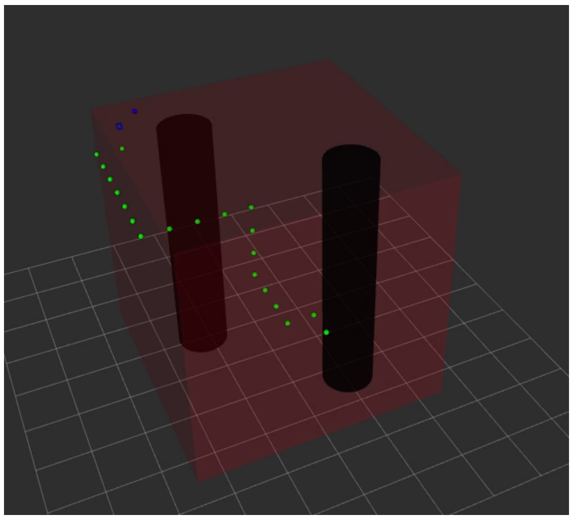
Training	Collisions	Found path
T1	16.1%	81%
T2	8.5%	94.2%

Table 3.13: Table showing a comparison of reliability results between trainings T1 and T2

Once these results were obtained, it was decided to continue with the proof of concept (POC) of this work. Using a Gazebo simulation environment for a UAV, the initial observations were taken, such as the initial position of the UAV after takeoff, the position of the obstacles in the environment and the target position. Subsequently, the ANN was feed forwarded with the weights obtained by T2 training to produce the path, which in the end, was delivered in the form of waypoints to the UAV. An image, showing the environment used for this POC can be seen in figure 3.10.



(a) Gazebo Simulation environment used for the POC.



(b) RViz Visualization environment used in this experiment

Figure 3.10: Figure showing a) the Gazebo simulation environment, where a UAV follows the path generated by the ANN, and b) the visualization environment in RViz, where the agent generates the path for the UAV.

CHAPTER 4

Conclusions and Future Work

4.1 Conclusions

In this thesis a Deep Q-Learning approach for path planning problems was presented. The experiments in this work go from low to high complexity systems. At the end, a discrete grid world represents the environment in which a UAV moves, this well abstracted representation of the environment simplified the path planning problem and solved it using Deep Q-Learning. Also, the learning was applied to a simulation environment using Gazebo open source simulation software and proved to be promising for a real world application.

Given the simplicity and generality of this environment representation, this algorithm can be implemented in other systems, as long as the robot can be represented as particle in its environment.

4.2 Future Work

As future work, this algorithm can be tested in more complex systems, like a 6 DOF robotic arm (see fig 4.1) or as mentioned above it can be tested in other systems, like for instance self driving cars or humanoid robots. Also, there is a large amount of new developments in the field of Deep RL, in the future state of the art ANN and learning policy's should be implanted using the approach presented in this work. From another point of view, the learning achieved in the experiments presented above, should be tested in real world systems.

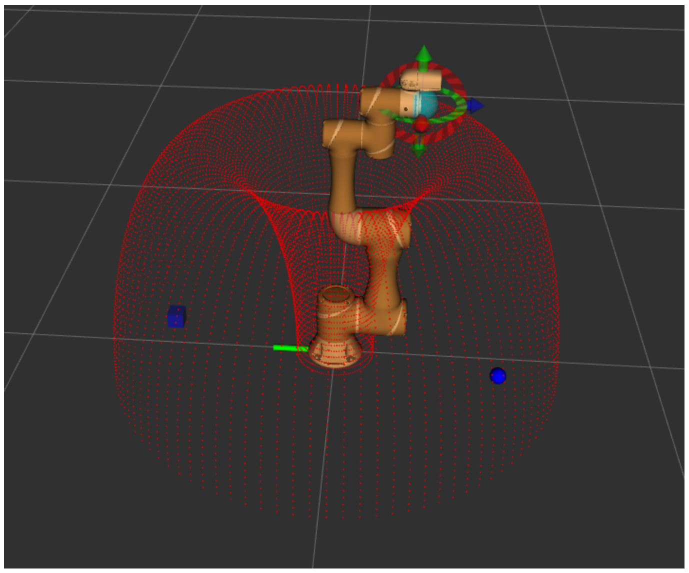


Figure 4.1: This figure shows one of the future work applications that can be done by continuing this work. A 6 DOF robotic arm with the environment that represents its collision free space can be observed as well, where the red dots represent the limits of this space, the blue point represents the initial position for the end-effector and the blue cube represent its goal.

CHAPTER 5

Bibliography

[1] J. Vásconez Hurtado, G. Kantor, and F. Auat Cheein, “Human–robot interaction in agriculture: A survey and current challenges,” *Biosystems Engineering*, vol. 179, pp. 35–48, 03 2019.

[2] K. Capek, *R. U. R.: Rossum’s Universal Robots*. Aventinum, 1920.

[3] G. C. Devol Jr., “Programmed article transfer,” Patent 2 988 237, June, 1961. [Online]. Available: https://www.freepatentsonline.com/2988237.html

[4] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, 1st ed. Springer Publishing Company, Incorporated, 2013.

[5] . Craig, *Introduction to robotics : mechanics & control / John J. Craig.* Reading, Mass.: Addison-Wesley Pub. Co.„ 1986, includes bibliographies and index.

[6] S. M. Lavalle, *Planning Algorithms*. Cambridge University Press, 2006.

[7] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[9] S. LaValle, “Rapidly-exploring random trees : a new tool for path planning,” *The annual research report*, 1998.

[10] K. Wei and B.-y. Ren, “A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved rrt algorithm,” *Sensors*, vol. 18, p. 571, 02 2018.

[11] K. Mi, J. Zheng, Y. Wang, and J. Hu, “A multi-heuristic a* algorithm based on stagnation detection for path planning of manipulators in cluttered environments,” *IEEE Access*, vol. 7, pp. 135 870–135 881, 2019.

[12] A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, 1959. [Online]. Available: https://doi.org/10.1147/rd.33.0210

[13] W. Mcculloch and W. Pitts, “A logical calculus of ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[14] C. C. Aggarwal, *Neural Networks and Deep Learning*. Cham: Springer, 2018.

[15] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, 1986.

[16] R. S. Sutton, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series)*. A Bradford Book, nov 2018. [Online]. Available: https://www.xarg.org/ref/a/0262039249/

[17] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Oxford, 1989.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013. [Online]. Available: https://arxiv.org/abs/1312.5602

[19] M. Mohammed, L. Kwek, and S. C. Chua, "Pick and place objects in a cluttered scene using deep reinforcement learning," *International Journal of Mechanical Mechatronics Engineering*, vol. 20, pp. 50–57, 09 2020.

[20] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," 2017. [Online]. Available: https://arxiv.org/abs/1708.04033

[21] J. Xu, T. Du, M. Foshey, B. Li, B. Zhu, A. Schulz, and others, "Learning to fly: computational controller design for hybrid UAVs with reinforcement learning," *ACM Transactions on*, 2019. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3306346.3322940

[22] A. Rodriguez-Ramos, C. Sampedro, H. Bavle, I. G. Moreno, and P. Campoy, "A deep reinforcement learning technique for vision-based autonomous multirotor landing on a moving platform," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1010–1017.

[23] G. Dissanayake, S. Huang, Z. Wang, and R. Ranasinghe, "A review of recent developments in simultaneous localization and mapping," in *2011 6th International Conference on Industrial and Information Systems*, 2011, pp. 477–482.

[24] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," 2017. [Online]. Available: https://arxiv.org/abs/1709.10489

[25] E. Marchesini and A. Farinelli, “Discrete deep reinforcement learning for mapless navigation,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 688–10 694.

[26] N. H. M. Sani, S. Phon-Amnuaisuk, T.-W. Au, and E. L. Tan, “Learning to navigate in 3d virtual environment using q-learning,” *Advances in Intelligent Systems and Computing*, 2018.

[27] L. He, N. Aouf, J. F. Whidborne, and B. Song, “Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 7491–7497.

[28] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente, and P. Campoy, “Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1024–1031.

[29] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[30] Stanford Artificial Intelligence Laboratory et al., “Robotic operating system.” [Online]. Available: https://www.ros.org

[31] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, “Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo,” 2016. [Online]. Available: https://arxiv.org/abs/1608.05742

[32] J. Matas, S. James, and A. J. Davison, “Sim-to-real reinforcement learning for deformable object manipulation,” 2018. [Online]. Available: https://arxiv.org/abs/1806.07851

[33] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis, “Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks,” 2018. [Online]. Available: https://arxiv.org/abs/1812.07252

[34] M. Gajamohan, M. Merz, I. Thommen, and R. D’Andrea, “The cubli: A cube that can jump up and balance,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 3722–3727.

[35] J. Xie, Z. Shao, Y. Li, Y. Guan, and J. Tan, “Deep reinforcement learning with optimized reward functions for robotic trajectory planning,” *IEEE Access*, vol. 7, pp. 105 669–105 679, 2019.