UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Tesi di Laurea Magistrale in Informatica

AI-based algorithms for drone routing in telecommunication scenarios

Relatore:
Ch.mo Prof.
Vincenzo DEUFEMIA

Candidato:
Gerardo De Rosa
Mat. 0522500722

Correlatore:
Ch.mo Prof.
Giuseppe POLESE

ANNO ACCADEMICO 2019/2020

For an end

is a new start.

CONTENTS

List of Figures **52**

ACKNOWLEDGEMENTS

Here I wish to briefly thank all the people that surrounded me and helped me in these last years.

Thanks to professors **Vincenzo Deufemia** and **Giuseppe Polese** for helping me with this project and the thesis with insightful ideas and dedication.

Thanks to my now colleague **Giuseppe Celozzi**, that as my company referent for Ericsson, has seen this project be born and grow, for being understanding and patient and also for providing me with the opportunity to become part of the Ericsson Company.

Thanks to my friends **Pasquale** and **Gerardo**, with which I shared part of my University life and still share moments of our daily life.

Thanks to my friends **Letizia** and **Corvo**, that listen to my non sense more often than it does them good, nearly every evening and night.

Thanks to my friend **Gianluca**, that has been by my side since High School, with which I shared all the University path and many different university projects.

Thanks to **my parents, my sisters and my nephews**, to be always there for me, motivating and tolerating me in the every day life.

Thanks to our nice cattoni **Lilith** and **Nano**, to sweeten my heart and allow me to caress them.

Thanks to my **lovvo Veronica**, for being my life and my all, without her I would have probably gone mad long time ago, thank you for being always there and

to always know what to say to cheer me up.

ABSTRACT

Drones are a technology growing increasingly with a huge potential in a variety of fields. Among this we can find the telecommunication sector, and of course the IT and AI ones. The purpose of the research presented in this thesis is to merge this two worlds to design and implement an algorithm that allows a drone to find an optimal path between a start point and an end point. The drone has the goal to reach the end point in order to give network support or detect interferences/faults. Two different AI-based algorithms have been considered, one based on Utility and the other on Reinforcement Learning (DQN). The model used to simulate the environment where the drone has to fly is the grid one, this led to choose the A* algorithm and then after some studies, to use an Utility function in place of the usual A* one, because the key thing to consider in choosing the path was not only the distance from the goal but also the time to reach it and the need to recharge the battery. This algorithm needs to know the environment and visit most of it, for this reason, it is also proposed an algorithm based on Reinforcement Learning that can also operate in an unknown environment. The Utility algorithm achieved excellent results, even with considerably large grids, while the Reinforcement Learning version the results where less exciting, mainly due to the difficulty of fine tuning the rewards to achieve the behaviour that one desires. Even if by using the Utility as the DRL reward function the number of times the algorithm finds a solution increased from 21 on 500 epochs to 200-250 on 500 epochs.

CHAPTER 1

INTRODUCTION

Path finding, autonomous driving and UAV/drones positioning, are among the most diffused AI algorithms in the recent years, this makes this field very crowded in terms of code and studies, every direction and new possibilities are carefully examined. Why is this field so relevant nowadays? Because it can used for a variety of purposes that make people life easier and safer; for instance we can think at self driving cars, or self driving boats that carry materials through difficult routes that could put men in danger. The baseline in these algorithms is the AI, thanks to it people can develop better solutions to these difficult problems. It combines the power of computation and algorithms with the environment state that the agent (protagonist of the AI) perceives through his sensors, in order to create a sort of behaviour that tells him what, when, and how to do things to achieve a certain goal.

A lot of AI applications are also found in the telecommunication field, for example to compute the best way to propagate the signal, to offer the right QoS depending on the zone, or like in this thesis, to compute an efficient path for a Unmanned aerial vehicle (UAV)/drone in order to allow it to give support to a network or detect faults/interferences in determined areas. Drones are indeed the principle actor in a lot of these scenarios, as we can see in the three main categories of study that regard AI based approach in telecommunications:

- *Interference detection*: this category includes all the problems that concern the interference between two or more network sources, usually private vs public.

- *Routing problem*: as said before, this is the field where the most relevant studies have been performed, they concern coverage, positioning problem, drones raid, and energy optimization.

- *Streaming* (image/video): under this category falls all the image/video elaboration related studies, that differ from one another for their purpose: obstacle avoidance, river monitoring, drone and crowded places recognition, and many others.

The goal of this thesis falls under the second category, as matter of fact, this thesis focuses on a path finding algorithm that differs from the existing approaches for the purpose, the methodology, and the technologies used. The combination of these three main features bring this study to be something innovative in the telecommunications field and also of interest for future research.

The main goal of the algorithm is to save enough battery of the drone while going from one point to another of the map in a certain amount of time. At the end point, the drone will provide support at the existing network or will provide supervision services to find faults or interferences' cause. In order to cover longer distances, some recharge stations can be found along the road, this are the places where the drone can fully recharge its battery, so that it can cover longer distances.

To achieve this result, it is used the A* algorithm together with the Utility function [1] to obtain a script that can find the best path to the goal performing the right amount of recharges at the designed spots. Successively, it is proposed a solution that exploits Reinforcement Learning [2] [3], and in particular Deep Q Network[1] , to make the algorithm work also in unknown environment and reduce the number of cell to visit in order to find the optimal path. In the creation of these algorithms, several technologies were used, the most relevant ones were: Visual Studio Code, the Tensorflow library for Machine Learning algorithms, and Python.

This work is organized in the following chapters:

- Second chapter: describes the state of the art of path finding algorithms and other studies done in the drone/telecommunications field.
- Third chapter: it introduces the proposed algorithms and the adopted technologies.
- Fourth chapter: it discusses the results achieved by the proposed algorithms in the experiments.
- Fifth chapter: it provides conclusions on this work and opens a door on the possible future work.

CHAPTER 2

STATE OF THE ART

As stated in the previous chapter, there are a lot of works concerning AI and drones in the telecommunication field; before putting myself to work on this project, I examined a large part of this studies. Here I will talk about some of them, also minding at the technologies involved in order to have a full picture of the matter at hand.

2.1 Ericsson research articles

Before talking about the papers and study I found online, I would also like to talk briefly of some interesting Ericsson research on drones and new technologies.

In This article, [4] the main subject is the job safety of the workers whose job is to survey really high antennas; Ericsson wants to use drones and AI algorithms in order to avoid the danger represented by having workers climb all the way up. It wouldn't be just an advance in security terms, because this algorithm can survey the places where the antennas are, much better and in detail, than a human can.

The article "How can we deploy drones in healthcare to save lives?" [5] is about hazard situations, in particular when there is an emergency situation and the drone is the fastest mean to deliver medical supplies, like defibrillators and such.

The prompt deployment of drones is already a reality thanks to 4g/LTE, but being able to be on the spot faster or to prevent this situation by using machine learning technology and IA, could really empower this mean. This will be even easier with 5G, that is capable of transfer more data and faster, so that the algorithms can work in a more fast and precise way.

This other article is about interference management [6], it talks about how with drones and 5G it will be possible to fly on suspect areas to detect possible anomalies. It will also be possible to create accurate 3D models of the network sites so that they can be explored without having to necessary go on the place. This is extremely good especially for the radio antennas situated in remote places.

2.2 Drones positioning algorithms

When I first approached the drone and UAV(Unmanned aerial vehicle) world, I had to perform a large research to understand what where the main interactions that this had with AI, in particular in the telecommunication field. Many of the study I've found are about the positioning of the drones and all the correlated problematics: choosing the right number of drones to deploy (raid optimization), algorithms to improve Coverage and Quality of Service, diminish the drone energy consumption ecc. I will talk about some of this papers, to also have a view on this issues.

In [7] the authors formulated the problem of positioning in a 3-D way, with the objective of maximizing the revenue of the network. They transformed the 3-D into an optimization problem and proposed a computationally efficient numerical solution to it; obtaining a script that can be used in many possible communication scenarios, including failure and congestion.

Another paper that studied this metrics, is the [8], where the efficient deployment of multiple UAV with directional antennas that provide coverage for ground users is analyzed. Also here the authors think in a three-dimensional way, they used circle packing theory, to determine the three-dimensional locations of the UAVs so that the total coverage area would be maximized while also maximizing

the coverage lifetime of the UAVs. With this method and various experiments on different sets of parameters, they discovered that to mitigate interferences and be compliant with the coverage requirements, the altitude and location of the UAVs had to be properly adjusted based on the beam-width of the directional antenna and the number of available drones, as we can see in the picture of Figure 2.1

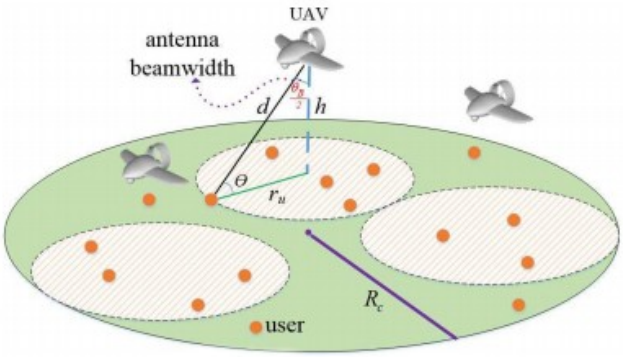

Figure 2.1: System model of [8]

Also in the paper of Rafaela de Paula Parisotto et al. [9] the problem of the positioning of drones is the main subject, but in this case the authors used the Reinforcement Learning to maximize the number of users covered by the drones while considering as constraints user mobility and radio access network. They managed to achieve good results that showed how the proposed algorithm could reduce interferences and disturbances for the users when compared to a fixed transmit power, all this by using only half of the drones usually necessary and lower average transmit power that could maintain the same coverage.

2.3 Drones image recognition algorithms

Another big part of the papers I encountered during my study, is represented by the ones about drones image and video recognition. The goals of the studies were different, from the obstacle avoidance, to the path identification, passing through drone or target identification. Like I did in the previous chapter, I will describe briefly some of the technologies and algorithms I encountered, but since the most of

this scripts use Convolutional Neural Network to elaborate images datasets, I will do a brief survey on the topic.

2.3.1 Convolutional Neural Network

CNN are a class of DNN that is used in the field of image recognition, in the last years they have achieved amazing performances on many complex visual tasks. They are used in image search services, self driving cars, drones and robots, video classification systems and so on. but they are often employed also in voice recognition and natural language processing tasks, even if, as I said before, they perform better in visual tasks.

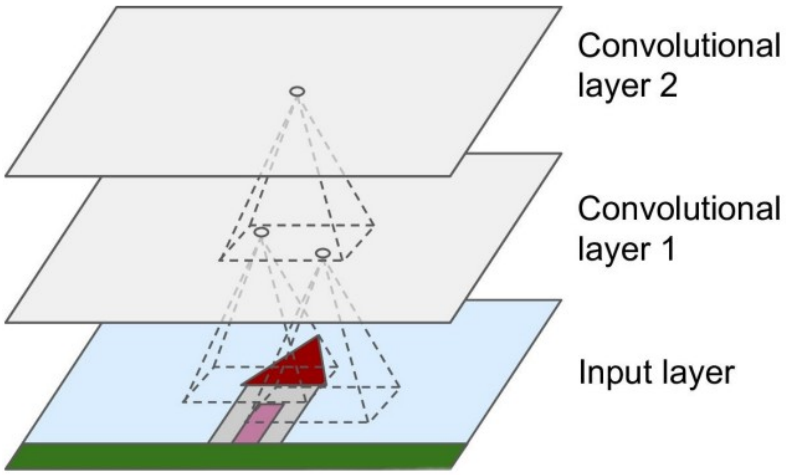

Figure 2.2: CNN layers with receptive fields [1]

CNN are based on convolutional layers, that contain neurons; this layers are placed one on top of the other; the neurons in the lower layer directly interact with the input image, each with the pixels in their receptive fields, equally, going in the layers above, each neuron is only connected with the neurons located within a small rectangle in the first layer, we can see and example in the figure 2.2. As one can think, in the drones field, CNN are used in path finding algorithms that use roads and road obstacles pictures, in order to elaborate a path and eventually as an algorithm of obstacles avoidance or object identification.

In [10], the authors used CNN to create a system capable of recognising road's

elements, they collected one thousand images of road objects, and developed an object detection system called faster R-CNN, composed of a fully deep convolutional layer and a Fast R-CNN detector. They performed Transfer Learning from a pre-trained deep learning model and also Fine-tuning to tweak the parameters in the best way; they then tested three different model in two different environment and achieved accuracy of 98% and 95%.

Zhilenkov et al in [11] studied a technique to create a system of drone automatic navigation based on a forest path. The main objective of the system was to find footpaths among trees which the drone could follow and in order to accomplish this, algorithms of artificial neural networks or CNN were used.

Figure 2.3: Output of the forest trail algorithm [11]

The algorithm structure was the following:

- First images and videos are recorded;
- A parallel optical unit processes the image;
- A convolutionary system of recognition and classification does the work;
- Finally a decision making system chooses the right direction.

We can see the result of the algorithm in the picture 2.3.

In [11] the aim of the authors, is instead, to present an urban intelligent navigator for drones using CNN. They captured one thousand images of six different street objects, those images were used as a dataset, that was then used as input of a machine learning algorithm that used Faster R-CNN like in [10]. Using different values for the parameters (initial learning rate and the batch-size) they created three different models, among this, the third one had the highest accuracy (98%) and could successfully detect and recognize all the objects at a specified location.

2.4 Path Finding algorithms

The state of the art of this particular branch of algorithms, has been under constant development in the last years, many scientist and developers approached the problem in many different ways; algorithms like the simple A* or Reinforcement Learning, DQN, Recurrent Network etc.

The article that inspired my work in the first place was one of this, it's an Italian study carried out in Turin [12]. It uses technique of Machine Learning and AI in order to secure the population from the flight of the drones; the aim of the algorithm is to process satellite and vector data, in order to build a risk map of the most crowded zone, to guarantee that the drones will avoid them and pursue instead more empty and safe places like threes, rivers and such. I will now present some of the other studies that I think to be relevant, but first I will talk about the most diffused technologies in the field of path finding.

2.4.1 A*

A* is an algorithm for path finding whose concept is rather simple: finding the shortest path between two places and it's based purely on the distance of each cell from the goal. This distance, that is an heuristic function, it's calculated as the sum of the distances from the start cell to the current one, plus the distance from the next cell to the goal, that is computed with an under value estimation.

$$f(n) = g(n) + ht(n), \forall ht(n) \leq ht*(n).$$

This algorithm at each step, evaluates where to move by calculating its heuristic for all the nearby cells and choosing the one with the minor value. The discarded cells are anyway kept in memory, because in the future it may be discovered that the chosen path is longer than the discarded cell estimated one, in that case the algorithm will go back and continue from there. The disadvantage of this algorithm is that it has to visit nearly always the most part of the cells to find the shortest path; apart from this, it is guaranteed to find the optimal shortest path. An example is shown in the picture 2.4.

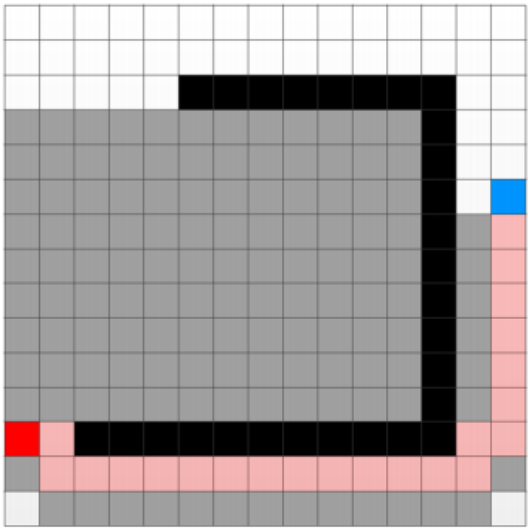

Figure 2.4: The grid cells examined during A* search from the red square to the blue square. The black squares are an obstacle. [2]

As Victoria J. Hodge and all say in Deep reinforcement learning for drone navigation using sensor data [2] "A global algorithm such as A* needs visibility of the whole exploration space, the drone would have to fly paths and backtrack to explore; also A* cannot cope with dynamic environments or next state transitions that are stochastic." For this reason it is not very used this days in path finding problems, furthermore other algorithms perform better and can be customized in a better way.

2.4.2 Reinforcement Learning

Some statements in this section are taken from the book of Aurelien Geron, Hands of Machine Learning with Scikit-Learn[1]. Reinforcement Learning is one of the most diffused field in Machine Learning, it is about the reinforcement of certain behaviours on top of others. Its objective is indeed to learn to act in a way that maximize its expected long term rewards: the agent operates in an environment and learns through trial and error. RL can be used in a variety of context, but the most diffused are the one concerning arcade games or teach robots how to behave. The algorithm that RL uses to determine its action is the policy, it can be a simple script or even a neural network that learns taking observations from the environment and how it reacts to the action the agent performs.

2.4.3 Markov Decision Processes

The interaction between the agent and the environment can be seen as a Markov Decision Process (MDP) where the interaction happens along a sequence of discrete time steps. An MDP is defined by the following components:

- A state space S, which is the set of possible states of the environment.
- An action space A, which is the set of actions that can be chosen by the agent.
- The dynamics of the environment, which is a probability distribution over next state s' and reward r, given the current state s and action a.

If both state space and action space are finite spaces, with a finite number of elements, the process is called a finite MDP. During interaction, at each time step t, the agent receives a representation of the environment state $S_t = s$ and on that basis selects an action $A_t = a$. During the next time step, the agent receives a scalar reward $R_t = r$ and the state of the environment is now changed to $S_t + 1 = s'$ which is jointly determined by the dynamics p. The outcome of these random variables, (s, a, r, s'), is referred to as an experience tuple. Bellman, that described MDP for the first time, found a way to estimate the optimal state value of any state s, he showed that if the agent acts optimally, then the **Bellman Optimality Equation**

2.5 applies; this equation says that the optimal value of the current state is equal to the reward it will get on average after taking one optimal action, plus the expected optimal value of all possible next states that this action can lead to.

$$V^*(s) = \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V^*(s')] \quad \text{for all } s$$

Figure 2.5: Bellman Optimality Equation. [1]

This equation leads directly to an algorithm that can precisely estimate the optimal state value of every possible state: first all the state value estimates are initialized to zero, and then they are iteratively updated using the Value Iteration algorithm 2.6.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s')[R(s, a, s') + \gamma . V_k(s')] \quad \text{for all } s$$

Figure 2.6: Value Iteration algorithm. [1]

Knowing the optimal state does not tell the agent explicitly what to do, so Bellman found a very similar algorithm to estimate the optimal state-action values, generally called QValues. The optimal Q-Value of the state-action pair (s,a), is the sum of discounted future rewards the agent can expect on average after it reaches the state s and chooses action a but before it sees its outcome, assuming it acts optimally after that action, it's written Q*(s,a). The Q-value iteration algorithm is shown in 2.7.

$$Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s')\left[R(s, a, s') + \gamma . \max_{a'} Q_k(s', a')\right] \quad \text{for all } (s, a)$$

Figure 2.7: Q-Value Iteration algorithm [1]

Once the optimal Q-Values are obtained, defining the optimal policy, is really important: when the agent is in state s, it should choose the action with the highest Q-Value for that state.

2.4.4 Q-Learning

The problem with the RL and MKP is that the agent initially has no idea what the transition probabilities are (it does not know T(s, a, s')), and it does not know what the rewards are going to be either. It must experience each state and each transition at least once to know the rewards, and it must experience them multiple times to have an estimate of the transition probabilities. The Q-Learning algorithm 2.8, is an adaptation of the Q-Value Iteration algorithm and it is a method to directly approximate the optimal action-value function, because since the target policy would act optimally, we take the maximum of the Q-Value estimates for the next state.

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha)Q_k(s, a) + \alpha\left(r + \gamma.\max_{a'} Q_k(s', a')\right)$$

Figure 2.8: Q-Learning algorithm [1]

Using this function in an algorithm and running it enough times it will converge to the optimal Q-Values.

2.4.5 Deep Reinforcement Learning & DQN

Q-Learning unfortunately does not scale well to large MDPs with many states and actions. The solution to this issue it to use a function that approximates the Q-Values using a not so high number of parameters, this is called Approximate Q-Learning. It was calculated for years with a linear combinations of hand-crafted features extracted from the state, but using deep neural networks really improved the performance of the algorithms, especially for complex problems, and it does not require any feature engineering.

A DNN used to estimate Q-Values is called a deep Q-network (DQN), and using a DQN to Approximate Q-Learning is called Deep Q-Learning and this kind of approach, in RL, is generally called Deep Reinforcement Learning. In order to make the training of the network stable, the authors of DQN proposed some ideas, the most used is the **experience replay** that helps to remove correlation between

experience tuples by storing them in a buffer. Instead of doing parameter updates sequentially with experience tuples as they are collected, random batches of tuples are sampled uniformly from the buffer to do updates.

2.4.6 RL and DRL algorithms

In the paper [13] RL is used for two main tasks: find UAVs flight path towards a designated mission area and avoid collision in stationary and mobile obstacles. The designed goal is not specified from the beginning, so that the UAVs can cover a certain mission area with a destination that can change during time. To reach the scope of this study, RL was used in combo with a new method to

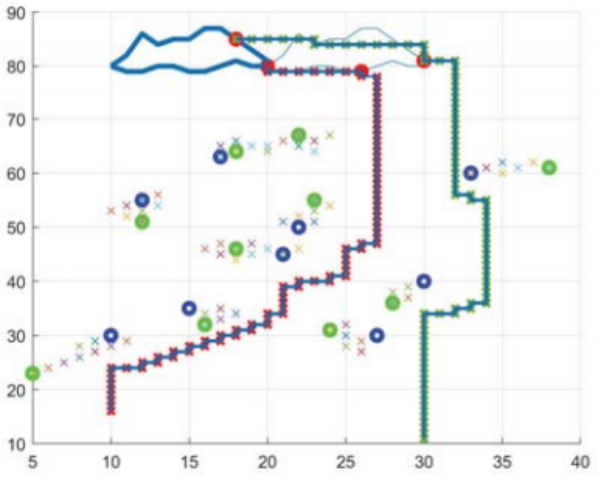

Figure 2.9: Simulation of the algorithm with time varying target. [13]

accommodate continuous state space for adjacent locations: instead of using function approximation methods the authors used Bellman equation as the action-value function, eliminating the need of exploring Q-value for each state. The assumption they made was that the Q-values for adjacent locations are correlated and hence slowly change over time during the exploration phase. The simulation results of the producted script show that success probability for this algorithm is about 80%, to drones to reach the destination without any collision, irrespective of the level of error variance. We can see an example of execution in 2.9

Moving forward, another study that uses DRL is the one of Aqeel Anwar et al.

[14]. It is about the autonomous navigation of a drone in a restricted environment, that can be a house or a room, via transfer learning (to have a litghweight alghoritm on board) of a value-based Deep Reinforcement Learning algorithm. The idea of the authors was to train a NN for an RL application in a variety of indoor environments collectively, and then use this knowledge with TL training with a smaller part of NN for similar application in a similar (but different/unseen) test environment.

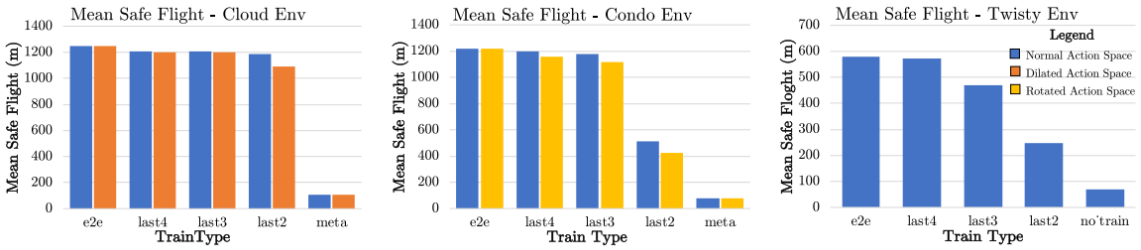

Figure 2.10: Mean safe flight (MSF) across different environment for different action spaces.[14]

To test the script, a library of 3D realistic meta-environments was manually designed using Unreal Game Engine and the network was trained end-to-end. To show robustness of the approach also variation in drone dynamics and in the environment characteristics were carried out and the approach was also tested on a real environment using DJI Tello drone reporting similar results. The MSF results are shown in the picture 2.10.

Another relevant drone navigation study is the one of Hodge et al in [2]. The authors conducted this study with the goal of developing a generic navigation algorithm that uses data from sensors on-board the drone to guide the drone to the site of the problem in dangerous and hazardous situation.

To implement the algorithm, it was used the proximal policy optimisation (PPO), deep reinforcement learning algorithm coupled with incremental curriculum learning and long short-term memory (LSTM) neural networks. The curriculum learning works starting with a simple task and gradually increasing the task complexity as it learns. This allowed the algorithm to gradually learn to navigate in complex environments with increasing obstacles. They also added a memory to the AI using a LSTM neural network that allowed the drone to remember previous

steps and prevent it from retracing them and getting stuck. To evaluate the script a Unity 3-D simulation environment was predisposed, and there were tested different configuration with different configuration of PPO and LSTM.

The results in figure 2.11 demonstrated that in general algorithm of PPO with LSTM length 8 performed better, except in very simple environments with few obstacles where a simple heuristic or PPO went straight to the problem and in very complex environments with many and complex obstacles where PPO with LSTM length 16 was better because it could retrace its steps in case the drones was surrounded by obstacles. It was also noted that the curriculum learning prevented the drone from wandering, assuring that it walked directly to the goal.

Algorithm	Grid size/number of obstacles							
	32/32	32/64	32/128	32/256	64/64	64/128	64/256	64/512
Heuristic	1986	1946	1717	1100	**1997**	1981	1873	1504
PPO	1993	1988	1930	1346	1994	**1984**	**1968**	1857
PPO_8	**1994**	**1992**	**1961**	1739	1992	1970	1950	**1858**
PPO_{16}	1985	1961	1944	**1866**	1944	1892	1811	1755

The total is taken across 2000 runs and higher values are better (fewer failures). The best (highest) value in each column is shown in bold

Figure 2.11: Accuracy, how many times the drone finds the goal.[2]

The next paper that I want to cite gave me some concrete ideas for my work, it is the one of Bayerlein et al. [15]. This study considers the problem of trajectory optimization for an autonomous UAV-mounted base station that provides communication services to ground users with the aid of landing spots. This spots alleviate the problem of the duration of same mission due to on-board battery budget of the UAV, which severely limits network performances. The authors used Q-learning to train a Neural Network that takes movement decisions for the UAV. This are made in a way that maximizes the data collected from the ground users while minimizes the power consumption by exploiting the landing spots. The results showed that the system intelligently integrated landing spots into the trajectory to extend flying time and that it was able to learn the topology of the network over several flying epochs.

The last study I will illustrate is actually a Thesis by Alex Hermansson Grogbeld [16] on the topic of RL to control a drone in a cellular network. The main core is to find paths between arbitrary locations like the low quality radio areas that are avoided at the cost of longer flight paths.

The reward function has simple parameters that he tweaked to modify the focus of the agent to put more emphasis on short paths or paths with higher radio quality. The agent he created uses a learning algorithm that combines Double Deep Q-Networks with Hindsight Experience Replay to handle the stochastic environment with multiple goals. To approximate the optimal Q-values a neural network was used and the experiences were collected using a Boltzmann exploration policy.

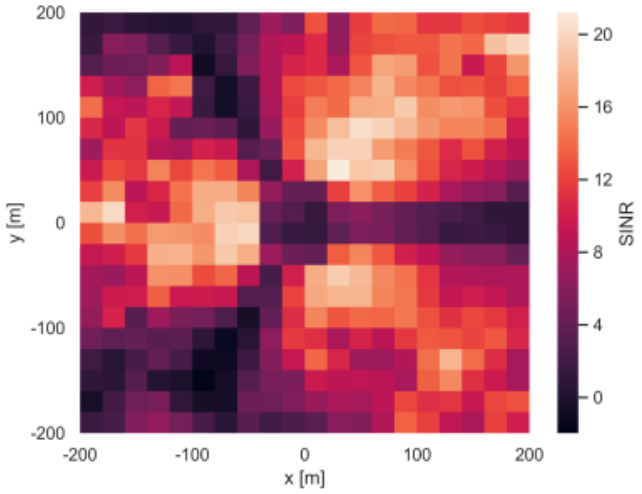

Figure 2.12: Visual representation of the two-dimensional environment, discretized into 20 × 20 bins. [2]

To test the environment and simulate the network, Ericsson provided a simulator that crafted measures of radio quality for an area surrounding one base station and with interference coming from six other base stations, or sites, in a close proximity, we can see an example in 2.12 The author studied three different scenarios:

- flight trajectories on constant altitude, with and without measurement noise in the radio quality.

- flight trajectories on varying altitudes with measurement noise.

In all the scenarios, simulation results showed that the agent successfully avoided low radio quality areas by taking longer flight paths, as of the goal of the study. The results also showed that the probability of flying through areas with low radio quality was reduced under the 75 percent and above the 62 compared to an agent that just flies towards the goal. He also noted that in 90 percent of the evaluated instances the flight paths, for all three scenarios, were shorter than two times the shortest possible path.

2.4.7 Other Path Finding algorithms

Here I will illustrate some of the studies that do not use the technologies that I talked about before but that face the routing problem in different ways. Kevin Dorling et al paper [17] is one of the most cited ones on this topic. It is about reducing the cost and time of making last-mile deliveries and responding to emergencies, so they talk about the vehicle routing problems (VRPs) in the drone delivery scenarios. The authors affirmed that existing VRPs studies are not good enough for drones, because they don't take into account battery and payload weight or multiple trip to deposit. They propose two multi-trip VRPs, both of them answer to the open issues illustrated before; one minimizes costs subject to a delivery time limit, while the other minimizes the overall delivery time subject to a budget constraint. The approach they used it's based on math, they derived and validated an energy consumption model for multi-rotor drones, demonstrating that energy consumption varies approximately linearly with payload and battery weight.

Thanks to this approximation they proposed a cost function that considers the energy consumption model and drone reuse, and applies it in a simulated annealing (SA) heuristic for finding sub-optimal solutions to the scenarios. The SA heuristic showed that the minimum cost has an inverse exponential relationship with the delivery time limit, while the minimum overall delivery time has an inverse exponential relationship with the budget. The result of the experimentation confirmed overall the importance of using drones and optimizing the battery size.

In [18] the authors instead, developed a system for conducting infrastructure inspections using drones that use GPS for autonomous flight control and estimate self-position through image processing when GPS cannot be used. The method involved detecting feature points from each ground image, that were taken directly from underneath the drone to estimate self-position, camera position and altitude, as well as mapping them.

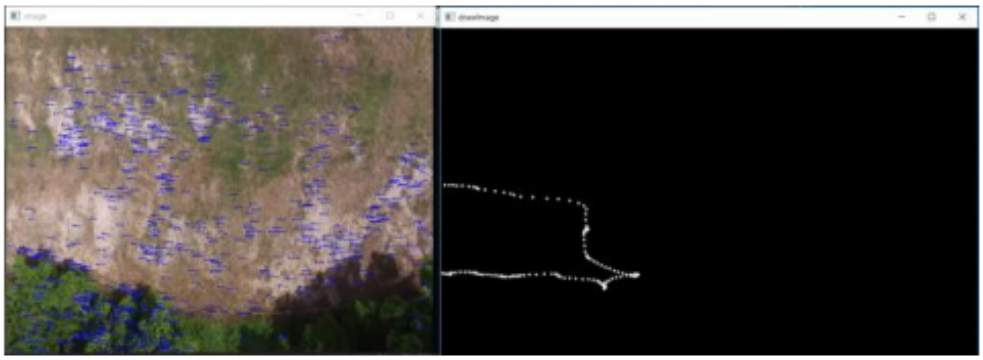

Figure 2.13: Result of the rendered trajectory. [18]

It was decided, that since the ground images taken aerially do not show texture clearly, the SURF algorithm, which enables detecting multiple features, must be used. After isolating the feature points, these have been matched in consecutive image frames thanks to Lucas-Kanade method and finally RANSAC was used to remove mismatches from the matching results. After also using the five-point to project a 3D model of the coordinates, the PnP problem was resolved based on it and the corresponding points in the image. In the end, to value the results, they reprojected the coordinates calculated trigonometrically, and obtained a rendered trajectory that was almost identical to the drone's flight path, we can see it in the figure 2.13.

2.5 Considerations

Among the technologies I described earlier, the ones with the better performance, adaptability and flexibility, are the ones that lie in the field of neural network, like DRL or CNN, this are still evolving fields, and if used and tweaked

in the right way, they can solve even really complex tasks. Instead, the math or heuristic based algorithms, can supply good results only in simple tasks and I don't think it's advisable to use them nowadays. That said, after studying the main field of AI in drones application, I choosed to pursue the path finding algorithms route, mainly because I sensed the field to be more appealing for me.

So after also consulting the paper [2] where there were listed different open issue on the drones matter, I decided I would elaborate a path finding script, but not a simple one, I decided that I wanted to create an algorithm that could make the drone cover longer distances or preserve more battery, by adding on the map, beyond that different kind of obstacles, recharge zones, where the drone could recharge itself, at the cost of losing more time and taking a larger path. In the next chapter I will describe in every detail the system and the scripts I created, I will then show the results I obtained and finally draw some conclusions.

CHAPTER 3

SYSTEM MODEL

The path finding system I propose in this study, as I mentioned also in the introduction chapter, is manly composed of two algorithms, one uses A* and utility, while the other uses RL and Utility as his policy. In this chapter I will thoroughly describe the various phases of development and what brought me to do the choices I've made.

3.1 Software and Libraries

To develop my project I used **VisualCode** as IDE, it is a really intuitive, easy and ready to use software developed by Microsoft, it offers a lot of extensions that help you coding, in fact it support a large variety of programming languages, and also usefull programs like git, that is already built-in.

As programming language, I used the classic **Python**, ver. 3.7.8, that is an object-oriented language, useful to develop distributed applications, scripting, numerical computation, system testing and machine learning.

In Python I used several different libraries, other then the classic math, random, numpy for the calculation, array and matrix, I used matplotlib for the plottig of the graphics and images, gym and tensorflow.

Gym is a library that allows you to operate with Reinforcement Learning, it provides you with atari games, classic RL and other kind of environments. It makes no assumption to the kind of Agent one uses and it's easy to learn. Once imported and declared the type of environment desired, it's ready to be used in the way that one wants to try out the RL alghoritm. It has a large community and a lot of developers enjoy creating new environments of different kind, in addition, it's easy to create a custom env so that one can use the fully potential of the tool to try out the own ideas. I did the same to test my DRL algorithm and to create my own environment.

Tensorflow is instead a more popular library, it's used for machine learning, it provides you many different methods to build your neural network, perform analysis, evaluate performances and so-on. It is used in a lot of different applications by many companies, among which we can also find Google. I used it to build the NN used for my DQN of which I will talk later on. The figure 3.1 shows the icons of the technologies i used.

Figure 3.1: Software and Libraries Used

3.2 System architecture

As I anticipated before, my work can be dived in two parts. The first part has been the development of an energy saver path toward a determined goal, using A* and Utility Theory. The algorithm I created performed really well, but as I said in the state of the art chapter, A* can operate only in known environments and it has to visit the most of it to find an optimal path. So I decided to use DRL, and I created a new algorithm that could work also in unknown environments. The

implementation part of my study can be divided in the following steps:

- Environment and Agent Creation
- A* and Utility Algorithm Creation
- RL Environment Creation
- DRL Algorithm Creation

3.3 Environment and Agent

The first thing that I decided was what kind of environment I would have used to run my scripts. I needed something that could abstract and represent the reality that I wanted to take in consideration in a simple but effective way. I opted for a grid environment like the ones I saw in some of the studies I described in the previous chapter. After choosing the environment, I populated each cell with a spot, of which I imagined six kinds:

- Recharge spots: happy islands where the drone can recharge its battery at the price of losing more time.
- Walkable spots: zones without any kind of obstacles or impediments;
- Minimum altitude specification: places where a minimum altitude is required to fly, like houses or buildings.
- Interference zones: spots where is present the signal of one or more antennas.
- Crowded zones: places where there is an overflow of buildings, people or other objects.
- Obstacles: places where the drone cannot go, like no fly zones, private properties and such.

Based on a determined percentage, different for every type of spot, I randomly inserted these in the grid. The main part of the grid is formed by walkable spots,

then we find the obstacles, while the less cell are the recharge zones; the others instead have more or less all the same number of spots on the grid. What differs the spots, apart from the presence and the meaning, is the value that each spot has on the map. This value is used by the Utility algorithm to determine the score of each spot, but I will explain this more in detail in the next chapters. In the figure below there is an example of the environment in a grid of 30x30, as it was at the begging of the study. The different colors represent the different kind of spots: yellow for the recharge zones, dark violet for the obstacles, violet for the interferences, light violet for crowded zones, and light blue for minimum altitude zones, as we can see in the figure 3.2.

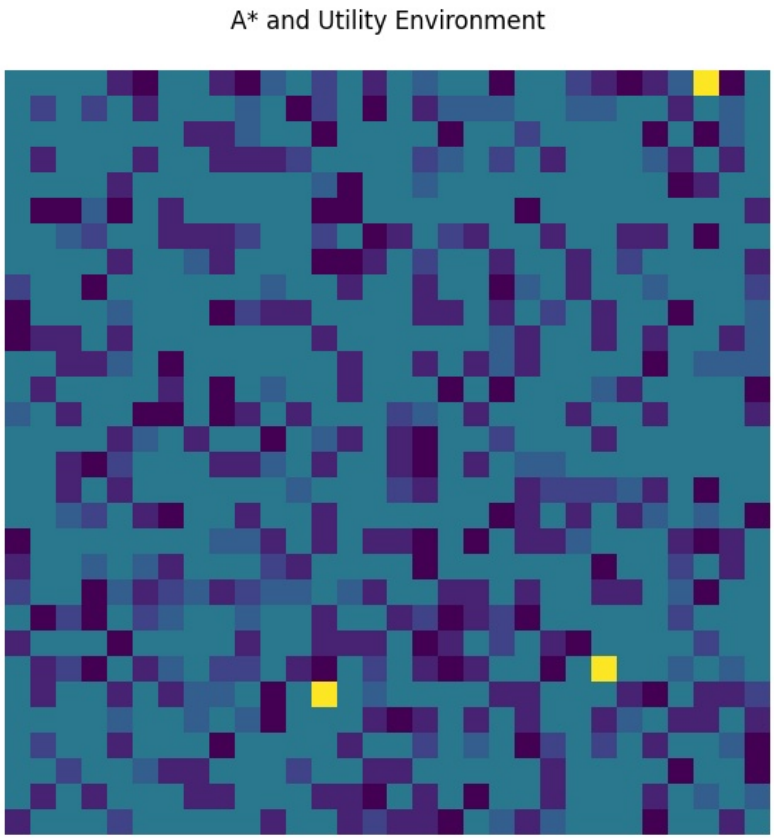

Figure 3.2: Environment

3.3.1 The Agent

In AI, an agent is an entity that performs actions in a specific environment in order to achieve goals. It uses observation through sensors to gain information about the environment and actuators to take actions in it. The behaviour of the agent is defined through specific algorithms or functions, and it changes to maximize its goal; in some cases it can also learn from experience.

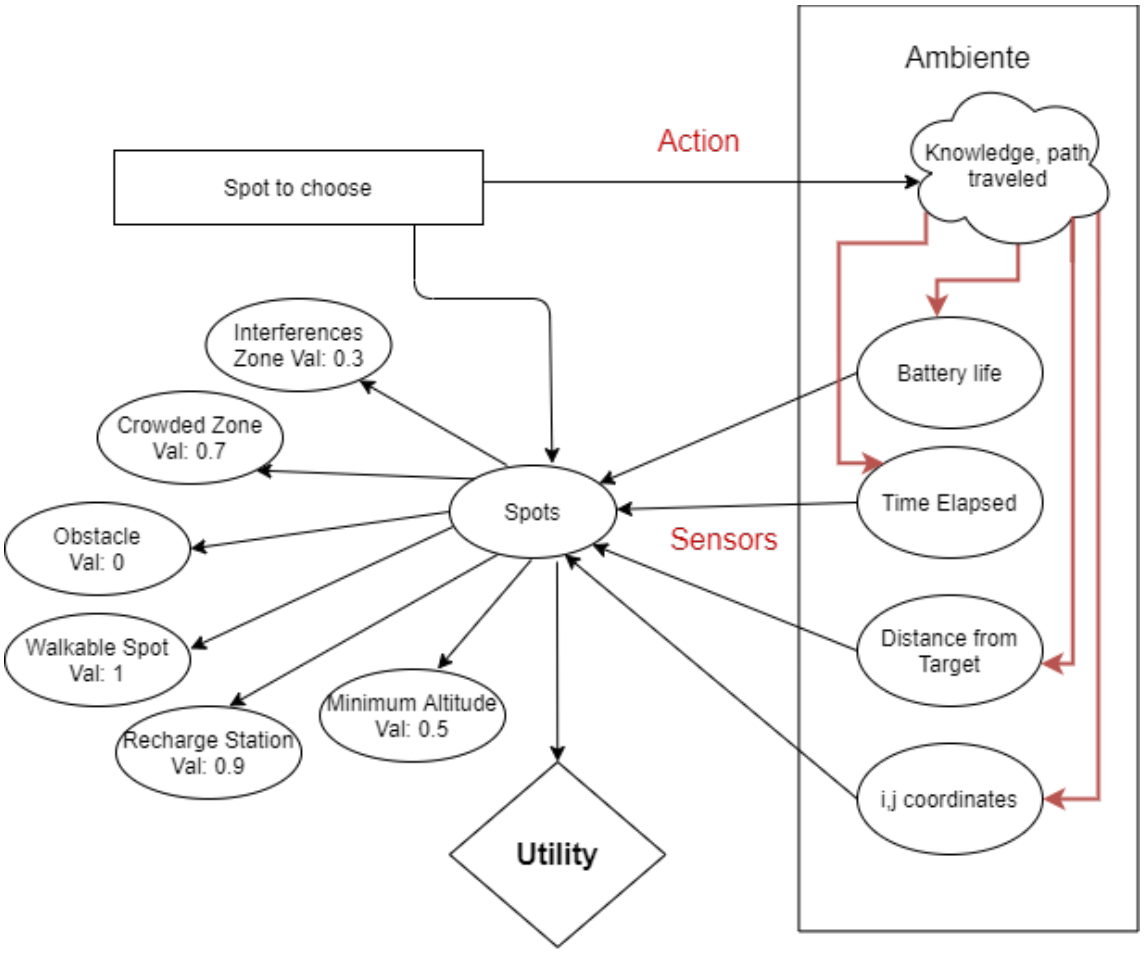

Figure 3.3: Agent utility feature

For what concerns the Agent/Drone, after the studies on the state of the art, I decided that it had to have five main attributes, that we can also see in the picture 3.3:

- Coordinates: i an j, to trace the grid spot on which the drone is on.
- Battery life: this variable keeps trace of the battery level;

- Time Elapsed: refers to the time from the start of the flight, it's important to trace it because, as I said, there is also a maximum time to reach the goal.
- Distance from target, this variable is fundamental for the Utility algorithm, in order to choose the best cell on which move.

The agent starts its trip at the cell (0,0), and it ends it at the proximity of the network he has to examine/help. At each iteration his attributes are updated with the function move, which purpose is indeed to update the values alongside with the decision of the algorithm.

3.4 A* and Utility Algorithm

In this section I will describe more in detail, the first algorithm I created and how. As I said, I used A* combined with Utility, but why? and how? To explain it better I will quickly explain what is Utility.

3.4.1 The Utility function

Generally speaking, Utility is a function in the field of AI, which purpose is to assign certain values to a set of actions that a system can perform, in order to let the system rank them to choose the most appropriate one for each case. It is usually applied in video games to develop characters AI, because in that contexts there must be an algorithm that creates a set of priority that change accordingly with the status of the agent.

In my case, Utility was used to rank the different spots available for movements, so I created an Utility function that fueled the A* algorithm, in order to keep the logic of A* but replace the distance function with an Utility one.

The function I created was conceived after different studies and tries to accomplish what I wanted. The baseline was that the battery, time and distance had to be the main factor together with the value of the spots examined. For me the main difficult has been to find the right way how to compute the right trade-off between time and battery, in order to guarantee that the drone would prefer the

recharge station spots only when its battery was really low and the time remaining was still enough.

$$UtilityFunc = (Weight[0] * Battery + Weight[1] * Time + 0.40 * Distance) * SpotVal$$

In order to achieve this, I acted on the weight that multiplies the value of the two features by using a proportion that balances the total weight, 0.60, between the two attributes.

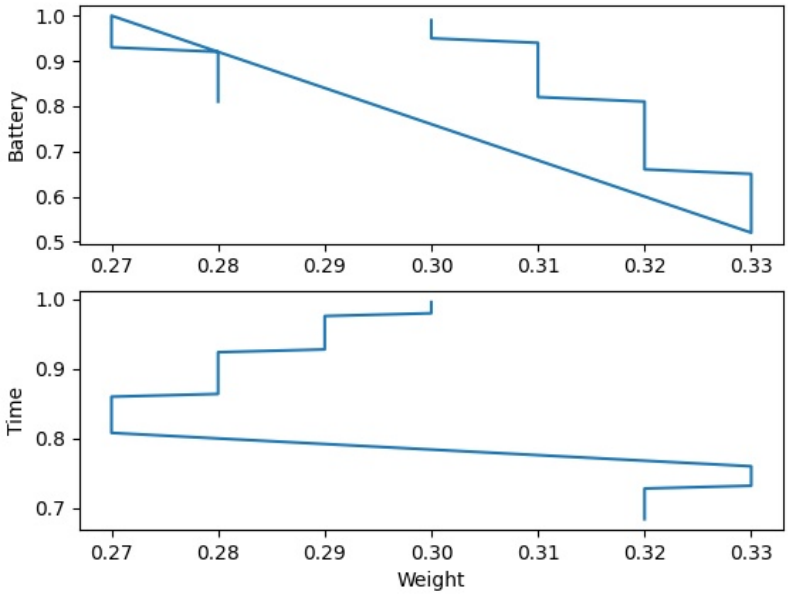

Figure 3.4: Time-Battery weight trade-off.

As the graphic shows in 3.4, both the weights starts at 0.30, then the battery one becomes higher, because it decreases faster than the time, that instead gets lower. At a certain point, the battery recharges itself and so its weight becomes lower than the time one, so that next time the drone will continue towards the goal and will not stop for a further recharge. The function that makes the drone lose battery and time and influences the changes in the weights was created ad-hoc by me: for every step the drone takes, in any direction, the battery value decreases of -0,01 while the time remaining decreases of -0.004. This is always true, except in the case of a recharge station, where the battery goes back to 1, being full charged, while the time decreases of -0.1.

3.4.2 The Algorithm

The first thing I do in the algorithm, it's generate the environment and put the different spots generated randomly inside of it. After this and the initialization of the drone, the main loop starts, inside of this the A* algorithm runs, and explores the environment in order to find the best path. At each iteration the nearby spots where the drone can go(it can move along all the axes if they are not walls) are evaluated; through the utility function the best spot is chosen for the movement, while all the others are kept in memory, like a classic A* algorithm, of which I described the basics in the second chapter. The Utility is fundamental in this part, because it keeps the drone from crushing or cross into the bad spots and allows it to recharge itself in order to cover longer distances and reach the goal.

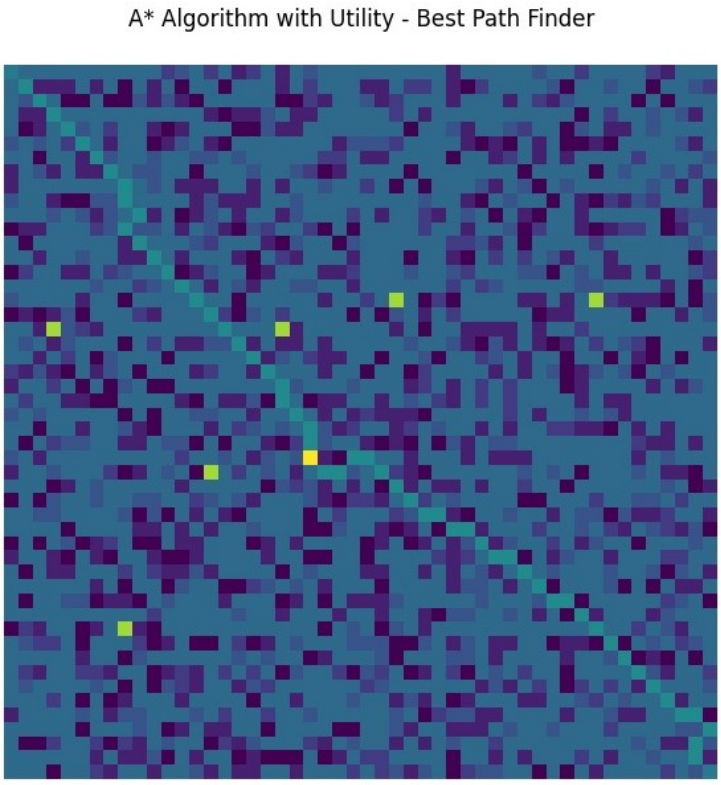

Figure 3.5: Example of A* Utility Algorithm result

The Utility is fundamental in this part, because it keeps the drone from crushing or cross into the bad spots and allows it to recharge itself in order to

cover longer distances and reach the goal.

Going forward the drone will continue exploring the env until it reaches the goal where the research script stops, and the final path that the agent choose is enlightened. Exiting the loop, the visualization function will finally plot the result of the execution, where we can see the path the drone took and the time it recharged itself and also the remaining battery and time will be printed out.

In the picture 3.5 an example of execution is showed, we can see that the drone stays on a steady path, avoiding obstacles, difficult spots and unnecessary recharges. Keep in mind that this is just an execution example, the test and the results will be discussed in the next chapter.

3.5 RL environment

I've already talked about the motivation that pushed me to transfer the algorithm from the A*/utility approach to the one of Reinforcement Learning in the previous section; The script of which I talked before, needed to explore the most of the environment and needed to know the value of each cell to find a path that satisfied the problem requirements. As I said in the state of the art chapter, RL is perfect to generalize this approach to unknown environment, that's why I choose to try and implement an algorithm that uses this technology.

The first thing that I had to do, was transport the environment that I used on the previous algorithm in the new one. To do so I studied the Gym library, of which I've discussed before; so I created my own custom environment, all I had to do was declare some specific functions and make some adjustment to the grid code I had already wrote. I also had to define an observation space, which is composed by the features that change at each step the drone takes and at which the rewards for every step are associated. These are really important because they are fundamental to value the goodness of the action and allow the RL algorithm to learn. In my case I choose to use just three of them because I thought and experimented that they were enough to guarantee an optimal learning:

- Battery;

- I coordinate;
- J coordinate;

Among the function I implemented, three were the most relevant:

- Initialization: in this function I initialize the environment, so I declare a new grid environment, a new drone, the space of possible action the drone can perform (8 because it can move in every direction) and the min e max value that the observation attributes can have.
- Step: in this function I program the reward for each action the drone takes; I've differentiated several cases, mainly to avoid the drone to get stuck in walls or to get away from the grid.
- Render: this function is important because it allows the graphical representation of the environment, usefull to also see other then just understand, what the drone did and what path did it learn.

The most important thing that is defined in this part of the algorithm is the reward function. The goodness of the algorithm relays mainly on it: it's the function that assigns a value to each action that the drone performs. I tried out different many different ones, but none of them performed particularly well; in the end I used the Utility function I created for the previous algorithm, but I put the spot value to 1 so that the algorithm could work in unknown environment.

3.6 DRL Algorithm

After preparing the environment, I've written the DRL algorithm, for which I used mainly the tensorflow library.

The structure is the basic one of a DRL algorithm, there is a class for the Q-Learning, one for the Buffered Reader and one for the DQNAgent; at the end there is the main loop where the training and the learning occur.

3.6.1 Q-Learning implementation

I've already talked about the theory part of Q-Learning in the State of the art chapter, so now I will only write about how I implemented it in my code.

The most important thing that happens in this class is the construction of the neural network I will use to predict the Q-values for each action and action*state pair, used to decide at each iteration on which spot the drone will move on. The NN I created is composed by three layers:

- The first one is the hidden layer that uses the ReLU activation function, it has as inputs a certain drone state and 128 units.
- The second layer is the one where we get the Q-values for each action, it has as input the hidden layer and the action size(8).
- the third layer it's where we get the value for Q(s,a) (state action pair); it's obtained by multiplying the states Q-value with a One Hot action vector I declared before in the code; then the sum is reduced to a single value by summing it across the columns.

Here I also defined the loss as the squared difference between the predicted q-state action and the q-target and the optimizer, for which I opted for the Adam one and set the learning rate to 0.01.

3.6.2 Replay Buffer

I used the Replay Buffer to perform an experience replay learning, of which I talked about in State of the Art chapter. I implemented two main function:

- The add function: simple appends experience to the buffer
- The sample function: that randomly choices some experience from the buffer tuple and then converts it in an individual list of state, action, next state and reward.

The main purpose of the replay buffer is indeed to use some of the agent's experience to elaborate its next move; in the train function, the method add and sample will be called, in order to add the experience at each iteration and take (sample) a certain number of previous experience from the buffer. In this way the Q-target for the function will be calculated from the list of rewards and the max of a list of the next states q-values.

3.6.3 DQNAgent

The DQNAgent it's maybe the most important part of the script, mainly because it's where the other class are initialized and there are two really fundamental function, the train and the get_action one.

The get_action function it's where it's decided the action that the drone will perform. Like first thing the state of the agent is used to take the q-values associated to it, then the greedy action is computed as the max of the state q-values.

To assure a fair exploration of the agent and that it doesn't remain stucked in local optimal, alongside with the greedy action also a random one is prepared, and thanks to a parameter called eps, one of the two action is chosen. I initialize the eps parameter to a value of 1, but this parameter is meant to go down with the passing of the iteration, so that after a certain number of iteration the algorithm, after learning based on his previous experience, should always choose the greedy action and be able to converge to a global optimal. In order to achieve this, every time an iteration ends, the eps value is multiplied by 0.995*, so that when it is confronted with a random value, to choose between the random action and the greedy one, slowly, the probability that the random one will be picked will converge to 0.

I adopted another trick to check if the next state that was gonna be choosed was gonna be outside of the grid. To prevent this from happening I simply simulated the step function with the original choosed action and checked the coordinates it returned, in case they were outside the range of the grid I hardcoded a step in a direction that allowed the agent to get away from the border.

The train function instead it's where the q-values are updated and the action correlated to the replay buffer that I described before happens. Also here I tweaked

a parameter, I'm talking about the gamma one. It's the discount rate that varies between 0 and 1; putting it simply the more close to 1 the value is, the less the algorithm will discount the future rewards. Having an high gamma parameters is good to make the model value future rewards worth equal or more than immediate rewards, while a lower one guarantees you the exact opposite.

I decided to have the initial gamma at 0.98, then when I manage to achieve the goal, it becomes 1.2, so that the later on steps I took towards the objective are valued more then the first one. I discovered that in this way the agent is more prone to retrace that steps in the later iterations, while still improving its performance. After the first time the goal is reached I decrease the gamma parameters by multiplying it to a fix value, and in n iterations it converges back to 0.98. The reason I do this, is that so that the agent doesn't get used to go around the map too much just to achieve higher rewards.

3.6.4 Algorithm Execution

At the end of the algorithm the main loop trains the agent for a determined number of times (epochs). Each time the algorithm cycles and the prediction table is updated, I also monitor some parameters like the total reward for each epoch and the times the algorithm manages to arrive at its destination. For each epoch, a clean environment is deployed and until it's done this operations are pursued:

- The action choosed by the DQNAgent is passed to the environment step function;
- The step function returns a list of 6 parameter:
  - State;
  - Action;
  - Next state;
  - Reward;
  - Done: 1 if the algorithm is terminated, 0 otherwise;

  - Info: can contain additional information about the execution of the step function

- This parameters are passed to the train function in the DQNAgent;
- The total reward variable and the state are updated;

After all the epochs are done executing, the environment and a graphic plot with some information are rendered, I will elaborate this better in the next chapter.

CHAPTER 4

RESULTS AND DISCUSSION

I will now perform some brief considerations and show the results of the scripts I discussed in the previously chapter. This chapter is organized in three sections, one for presenting the results of A* Utility algorithm, one for the DRL/Utility one, and finally, a comparison between the two algorithms is discussed.

4.1 Evaluation of A* and Utility algorithm

After some parameters tweaking, This algorithm has always performed really well even on really large grids like 100x100. To try it out I re-designed some of the environment aspects, in the case the drone had the goal of giving support to an existing network. I placed three drones/antenna on the map with the signal beam as a parameter that represents the number of cells that the signal of that specific antenna reaches. The goal of the drone in this case is to get to the place where the fourth antenna should be. Once arrived it starts providing his signal too.

The environment on which the agent has been tested are random generated, like also the position of the other drones/antennas. Every time based on the position of the top antenna, the position of the others it's computed.

In 4.1 we can see an example of the execution on a 100x100 grid. We can

notice how the three antennas overlap a bit on the borders, but this is a normal thing in networks, because to offer enough coverage some already covered areas may be afflicted too.

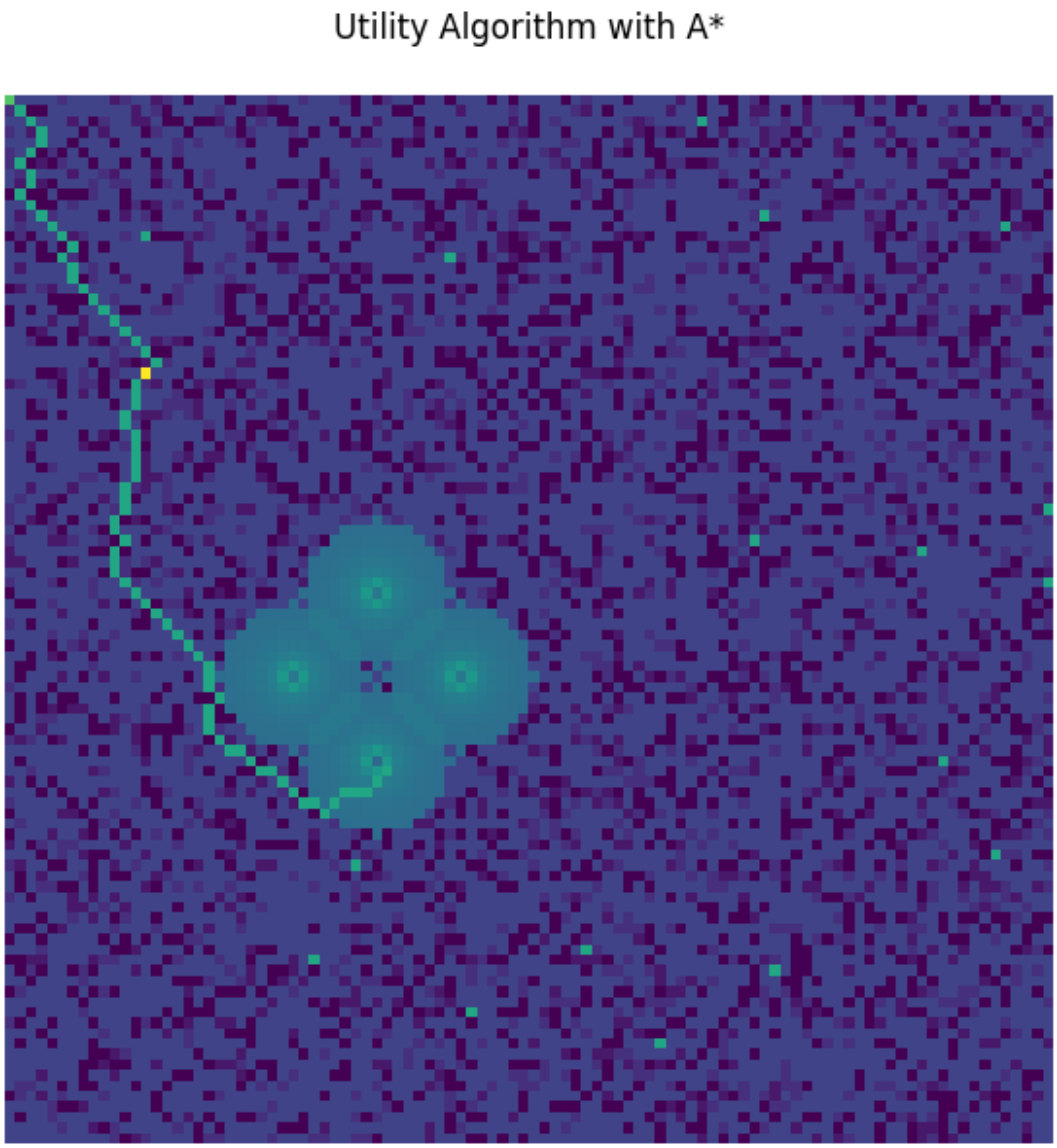

Figure 4.1: A* Utility Algorithm result with antennas.

The fourth antenna is the drone who got there thanks to the algorithm, as we can see it has only performed one recharge at a certain point. In this algorithms the drones nearly always finds the goal, when it doesn't succeed it's usually because the random generated obstacles cover every possible path, or when the recharge stations are more far away then the actual target and thus unreachable, making the battery

of the drone die.

The advantage of using this algorithm is that it's fast and performs well also on grid of considerable dimensions. The downside is, though, that it has to explore nearly every cell to find an optimal path and that if the kind of spots are not specified it cannot perform well.

4.2 Evaluation of DRL whit Utility as reward function

Crating a RL algorithm it's not easy as this article reports [3]. The main issue is that the RL purpose is to maximize it's rewards, and it nearly never means that this will coincide with your goal, because most of the time the algorithm will find different ways to accomplish his scope different from what you believe to be the reasonable behaviour.

So the most important thing that I did after creating the DRL script and the RL environment, was to try out different reward function that could provide the right values for what I meant to do. Other then that I also tried out different configuration of some parameters like the learning rate and the gamma, lambda ed epsilon value, of which I talked before.

After a really long session of parameter tweaking, I decided to use the Utility function as the reward one, but in order to do that I removed the parameters that indicated the value of each specific cell, that usually was multiplied at the end of the function. The Utility function is used in a different way from the one I used in the another algorithm; in fact it's only used to provide a reward, after that the direction of the movement it's already decided.

By using this configuration I managed to pass from 21/500 success on a grid of max 40x40 to a success rate of 200-250/500 on a grid of 60x60, even if the script also works on at least 70x70 grid. The success of the algorithms depends on the correct exploration of the grid world by the agent, if it manages to find the goal once, most likely it will find it again and in the meantime improve its performance. If this doesn't happen the script will end and the drone will never make it to the

goal. The main challenge I had to face with this algorithm, is the creation of the best reward function possible that would make the drone learn that it has to pass on recharge station spots in order to reach the most far cells and it also has to avoid them when there is no need at all for recharge.

I want to point out though that all the other algorithms of the papers I analyzed, that use RL and the grid environment, are tested on small grid of in average 30x30 [16] [2], while my algorithm can run also on larger ones and in addiction once it has find the path, it performs equally on all dimensions grid: meaning that it can reach the destination the same number of times independently from the grid size.

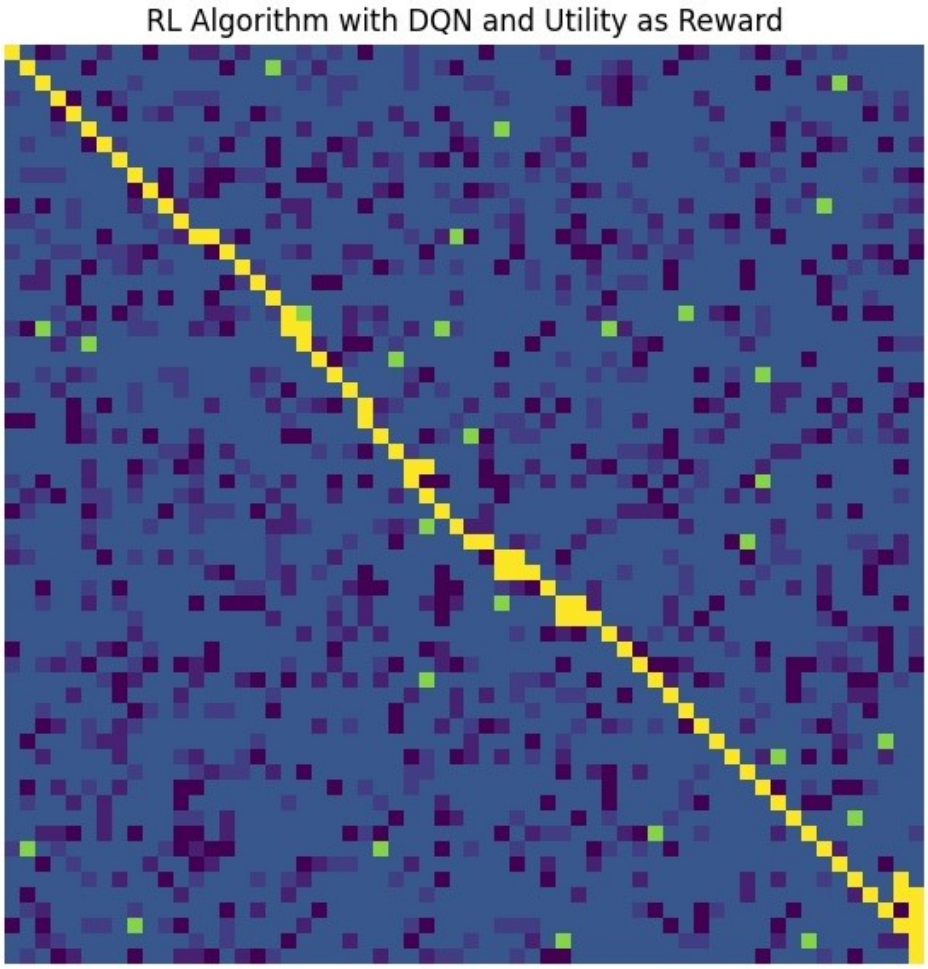

Figure 4.2: DRL with Utility Reward Function

As we can see in the picture 4.2, the drone manages to reach its destination

and performs just one recharge, while all the other recharge spot are instead avoided. We can see that the algorithm is not flawless and sometimes it passes on some cells that are not necessary, unfortunately with DRL there is no way to accomplish a cleaner result; especially with near cells stuff like this can happen really often.

As we can see in the plot 4.3, the epsilon parameter goes down while the epoch score increases, like it should be; this means that the drone starts to converge to a certain score that is the global optimal and there is no need for the epsilon value to keep it from getting stuck in the local optimal, generating random actions.

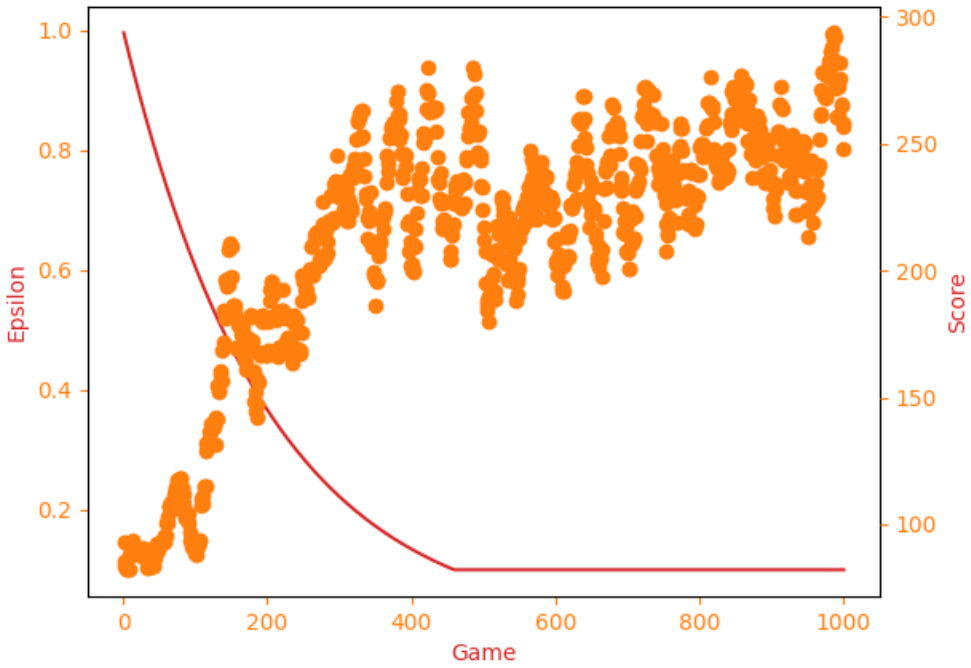

Figure 4.3: Reward x EPS x Epochs plot

4.3 Comparison of the Algorithms

From the results I presented it is evident that even if the DRL performs nicely, the A* and Utility algorithm is more precise and fast then the other one. So why should someone use the DRL approach? Because it works also in unknown environment and that is a fundamental thing to use this algorithm in more realistic

use cases. In addition the DRL algorithm scales better with the growth of the grid and maybe with some other trick or a different reward function it could also improve its performance.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

As I said in the introduction, drones and AI are a really actual topic, as we have seen in the state of the art section, nearly all the paper are dated max 4 years ago, it means that the field is being explored the most in this very moment. I'm really satisfied of the research activity I conducted, because I managed to explore a vast majority of the literature and learn many important things that I then used in the implementation phase.

I managed to achieve the results I proposed myself with success, I created and proposed two algorithms that have the objective of finding a path for a drone that must be the most energy efficient and fast as possible. These requirements are mandatory because of the destination place, that can be a network where the drone has to give net support, or a network that may suffer from a fault or interferences and needs to be checked out; in both these cases the battery and the prompt intervention are fundamental factors.

The first script I created uses A* and Utility to create the path and performs really well on grid of various dimensions; the other one instead uses DRL, DQN and the Utility as a reward function [1], it performs worse then the other one but this works also on unknown environment.

I'm satisfied about this results, even if I hoped to find a better DRL reward

function, but it was not possible considering the difficult of the task and that the DRL is hard to tame.

5.1 Future Work

Thanks to my Ericsson referent in this study, Giuseppe Celozzi, I had the chance to learn something also on the purely network topic, it was really important for me and the development of my idea and application. He also supplied me with different Ericcson research articles, some of which I've spoken of in the second chapter, thanks to them I've elaborated some nice ideas that I will now briefly present:

- Translate this algorithm in the sea environment, to treat the problem of the UAVs that have to move really consistent packages via sea: I would have to consider other variables, mainly the meteorologic ones;
- Transform this algorithm to adapt it to a real world environment, taking satellite images and maps and then even deploy it on a real drone to test it out;
- Use the DRL approach to create an algorithm that could fly over the network with interferences or faults in order to discover them, but being careful to not being too close to the network to not become another source of interferences.
- Try to find other strategies to improve the DRL algorithm performances.

BIBLIOGRAPHY

[1] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017. ISBN: 978-1-491-96229-9.

[2] V.J. Hodge, R. Hawkins, and R. Alexander. "Deep reinforcement learning for drone navigation using sensor data." In: *Neural Comput & Applics* 33 (2021), pp. 2015–2033. DOI: https://doi.org/10.1007/s00521-020-05097-x.

[3] Alex Irpan. *Deep Reinforcement Learning Doesn't Work Yet*. https://www.alexirpan.com/2018/02/14/rl-hard.html. 2018.

[4] Andres Torres. "Afraid of heights? Drones, AI and digitalization to the rescue!" In: (2020). DOI: https://www.ericsson.com/en/blog/2020/3/intelligent-site-engineering/.

[5] Mal Raddalgoda. "How can we deploy drones in healthcare to save lives?" In: (2020). DOI: https://www.ericsson.com/en/blog/2020/2/drones-in-healthcare-canada.

[6] Johan Torsner et al. "Interference management in 5G with drones". In: (2020). DOI: https://www.ericsson.com/en/blog/2019/9/interference-management-5g-drones.

[7] R. I. Bor-Yaliniz, A. El-Keyi, and H. Yanikomeroglu. "Efficient 3-D placement of an aerial base station in next generation cellular networks". In: *2016 IEEE*

International Conference on Communications (ICC). 2016, pp. 1–5. DOI: 10.1109/ICC.2016.7510820.

[8] M. Mozaffari et al. "Efficient Deployment of Multiple Unmanned Aerial Vehicles for Optimal Wireless Coverage". In: *IEEE Communications Letters* 20.8 (2016), pp. 1647–1650. DOI: 10.1109/LCOMM.2016.2578312.

[9] R. de Paula Parisotto et al. "Drone Base Station Positioning and Power Allocation using Reinforcement Learning". In: *2019 16th International Symposium on Wireless Communication Systems (ISWCS)*. 2019, pp. 213–217. DOI: 10.1109/ISWCS.2019.8877247.

[10] I. G. M. I. Moteir et al. "Urban Intelligent Navigator for Drone Using Convolutional Neural Network (CNN)". In: *2019 International Conference on Smart Applications, Communications and Networking (SmartNets)*. 2019, pp. 1–4. DOI: 10.1109/SmartNets48225.2019.9069781.

[11] A. A. Zhilenkov and I. R. Epifantsev. "System of autonomous navigation of the drone in difficult conditions of the forest trails". In: *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. 2018, pp. 1036–1039. DOI: 10.1109/EIConRus.2018.8317266.

[12] M. Mozaffari et al. "From shortest to safest path navigation: an AI-powered framework for risk-aware autonomous navigation of UASs". In: (2020). DOI: https://poliflash.polito.it/awards/e_del_politecnico_l_unico_progetto_italiano_premiato_da_amazon.

[13] S. Islam and A. Razi. "A Path Planning Algorithm for Collective Monitoring Using Autonomous Drones". In: *2019 53rd Annual Conference on Information Sciences and Systems (CISS)*. 2019, pp. 1–6. DOI: 10.1109/CISS.2019.8693023.

[14] A. Anwar and A. Raychowdhury. "Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning". In: *IEEE Access* 8 (2020), pp. 26549–26560. DOI: 10.1109/ACCESS.2020.2971172.

[15] H. Bayerlein, R. Gangula, and D. Gesbert. "Learning to Rest: A Q-Learning Approach to Flying Base Station Trajectory Design with Landing Spots". In: *2018 52nd Asilomar Conference on Signals, Systems, and Computers*. 2018, pp. 724–728. DOI: 10.1109/ACSSC.2018.8645103.

[16] ALEX HERMANSSON GROBGELD. *Network Drone Control using Deep Reinforcement Learning*. School of Electrical Engineering and Computer Science, 2019.

[17] K. Dorling et al. "Vehicle Routing Problems for Drone Delivery". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47.1 (2017), pp. 70–85. DOI: 10.1109/TSMC.2016.2582745.

[18] H. Kinjo et al. "Infrastructure (transmission line) check autonomous flight drone (1)". In: *2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. 2017, pp. 206–209. DOI: 10.1109/ICIIBMS.2017.8279694.

LIST OF FIGURES