SAPIENZA
UNIVERSITÀ DI ROMA

Multi-UAV reinforcement learning with temporal and priority goals

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Alessandro Trapasso

ID number 1597997

Thesis Advisor

Prof. Luca Iocchi

Co-Advisor

Prof. Fabio Patrizi

Academic Year 2019/2020

Thesis defended on 31th March 2021
in front of a Board of Examiners composed by:

Prof. Alberto Marchetti Spaccamela (chairman)
Prof. Giorgio Grisetti
Prof. Luca Iocchi
Prof. Giuseppe Di Luna
Prof. Simone Scardapane
Prof. Fabrizio d'Amore
Prof. Christian Napoli

Multi-UAV reinforcement learning with temporal and priority goals
Master's thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: trapasso.1597997@studenti.uniroma1.it, ale.trapasso8@gmail.com

Abstract

Unmanned aerial vehicles (UAVs) are becoming more and more in demand, their use initially spread for the execution of military missions, now it is becoming increasingly popular in the civilian field. UAVs are involved in the most exterminated missions. In this thesis, we address the problem of the risk of conflicts between UAVs. Collision avoidance is a problem that also affects the BUBBLES project, a project funded by the European community. We abstractly define the problem and then solve it by introducing a no-interference function that allows us to solve the possible conflicts between agents, so, between UAVs. The main techniques used to solve the problem are Reinforcement Learning and $\text{LTL}_f/\text{LDL}_f$ temporal logics.

Dedicated to
my parents

Acknowledgments

..

Contents

Chapter 1

Introduction

In this descriptive chapter, a general overview will be made: on the reasons for which I decided to work on this thesis project, on the operational scenario, on the definition of the problem, and on the techniques used to solve it.

1.1 The importance of *UAVs* in the near future

In recent years, drones have enjoyed considerable success, their market is constantly growing, both in the military and in the civil sector. *UAVs* have been around for many years, even the first *UAVs* designed date back to World War I, when both the United States and France were working on unmanned automatic airplanes. Since then great strides have been made, *UAVs* are used in different fields, from agriculture to video surveillance, from the delivery of network services to the delivery of medical supplies, food, or other goods. This last field is the one that intrigued me most, in particular the delivery of medical supplies of any kind.

1.2 Motivation

The interest in *UAVs* is constantly growing, in the coming years they will be used in many fields and many jobs in the civil and military sectors will be replaced by drone technology. Before that happens, however, there are several engineering challenges to be solved, one of these arguably the most important being safety. By safety we mean the minimization of the risk of collision and the analysis of potential conflicts based on the position of the trajectories in space.

This thesis project was born for the personal interest in *Machine Learning* and in

particular, for the *Reinforcement Learning* algorithms applied soon in which *UAVs* will be able to manage and share airspace in total safety according to temporal logic and priority.

Minimizing the risk of collision is covered extensively in the *BUBBLES project*. The *BUBBLES project* is a large project funded by the *European community*, coordinated by the *Universitat Politècnica de València* with the participation of the *DIAG, Sapienza University of Rome* and other public and private bodies. The main objective of the *BUBBLES project* is to create *Artificial Intelligent (AI)* algorithms to calculate the collision risk of *UAS* leading to minima and separation methods. The scenario of this thesis project was inspired and then remodelled by an operational scenario present in the *World package* 3 *(Catalog of Generic ConOps)* of the *BUBBLES project*.

1.3 Proposed Solution

Minimizing the risk of collision is a fundamental requirement to carry out *Multi-UAV* missions safely, this requirement is more important than efficiency and must be guaranteed. In this thesis as in the *BUBBLES project*, the minimization of the collision risk is the primary requirement. Another fundamental objective of this thesis is the management of missions according to their priority. Hence, each *UAV* must reach its goal in the shortest possible time based on the priority of the task assigned to it, ensuring *safety*.

To solve this problem, *Reinforcement Learning* algorithms are used which through the use of rewards allow each *UAV* to reach its goals and to manage priorities a Restraining Bolt is used for each agent, which imposes the restrictive specifications using the linear temporal logic on $\text{LTL}_f/\text{LDL}_f$ finished traces.

To achieve this goal, *Reinforcement Learning* algorithms were used: *Q-Learning* and *SARSA*, while the *Restraining Bolt* is specified logically using linear time logic on finite traces $\text{LTL}_f/\text{LDL}_f$ over a set of high-level symbolic features (De Giacomo et al., 2020). The *Restraining Bolt* has the function of imposing restrictive specifications to limit the agent's actions to a series of desired behaviours. This limitation is specified by the $\text{LTL}_f/\text{LDL}_f$ formulas, which in our case will specify to the agent the temporal and priority goals.

1.3.1 Scenario Characteristics

As it was previously said, the application scenarios in which *UAVs* can be used are very many, the scenario that is examined in this thesis project concerns the delivery of medical supplies between hospitals in a metropolis.

For our purpose, eight hospitals in the city of Rome have been represented on the grid, each hospital can request medicines, organs, vaccines, blood from one of the other seven hospitals in the city. There are three *UAVs*, each *UAV* is located in a random point on the map and has the aim of fetching the required element e.g. from hospital 1, and deliver it in the shortest possible time, safely, to the hospital 2, taking into consideration the priority of its mission and that of the other *UAVs*. The scenario, problem definition, and proposed solution will be discussed in more detail later.

1.4 No-interference reward

In chapter 4.4 we will introduce the theoretical problem of the interference between policies. We will provide an abstract definition of the problem, and then focus on an instance of the problem itself, which will lead us to the definition of the thesis scenario. We will introduce the concept of priority and explain how it will be managed, both as regards the goal associated with the UAV, and as regards the management of the order of hospitals visited according to a temporal priority $\text{LTL}_f/\text{LDL}_f$. In solving the problem we will better understand the no-interference reward function and how it can help us eliminate any interference between the agents' policies. In the section dedicated to the case study, we will understand how the trajectories generated by the optimal policies of each agent will be useful for defining our no-interference reward function.

1.5 Thesis structure

The thesis is structured as follows:

- In the chapter 2, we will understand some theoretical notions that will be useful later on. We will see in detail what Reinforcement Learning is and what temporal formulas are through theoretical notions and examples. We will start at LTL and then arrive at LTL_f and LDL_f, which we will use in our work together with Reinforcement Learning algorithms. We will also understand the objectives of the BUBBLES project.

- In chapter 3 we will describe the OpenAI Gym toolkit that we will use to create our environment and we will see two works that inspired this thesis project.

- Chapter 4 the chapter of defining the problem, in this chapter, we will abstractly define the problem, we will understand how to manage priorities and what will be

the solution we propose. We will also go into the detail of the case study of this thesis.

- The chapter 5 describes how the environment was created and which parameters can be changed. It will be possible to better understand how the Restraining Bolt manages time priorities and how the "no-interference reward" will be implemented.
- Chapter 6 presents the results obtained empirically from the implementation, we will understand what the evaluation criteria are and compare the IV cases studied.
- The thesis is concluded in Chapter 7. This chapter summarizes also the achievements of the thesis and discusses future works

Chapter 2

Related Work

2.1 Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning where Agent learning occurs through rewards and punishments (Sutton and Barto, 1998). It differs from supervised learning and unsupervised learning. In supervised learning input is provided as a labelled data-set, a model can learn from it to provide a problem outcome. Problems can be of two types: problem classification, regression problems. Unsupervised learning is completely the opposite of supervised learning. This algorithm has as its primary purpose to explore the underlying patterns and predicts the output. In RL the *Agent* is the learning entity, the *Agent* will try to maximize the observed reward (Ng et al., 1999) by interacting with the environment through actions. However, we need to define some very important concepts in RL: *Agent*, *environment*, *state* and *action*. The *Agent* performs actions in the environment and receives *observations* and *rewards*. The *environment* is the overall representation of which the agent interacts. The *Agen* is not part of the *environment*. The *environment* receives action and issues remarks and reward. The state describes the current situation and determines what will happen next. When the Agent has a partial view of the state, it is called observation. Action is what an agent can do in any state. Actions allow the agent to interact with the environment. Each action performed by the agent produces a reward from the *environment*. The decision of which action to choose is made by *policy*. The *policy* is denoted by π and is the solution to an RL problem, the *policy* determines which action should be executed in a given state to maximize the long-term reward.

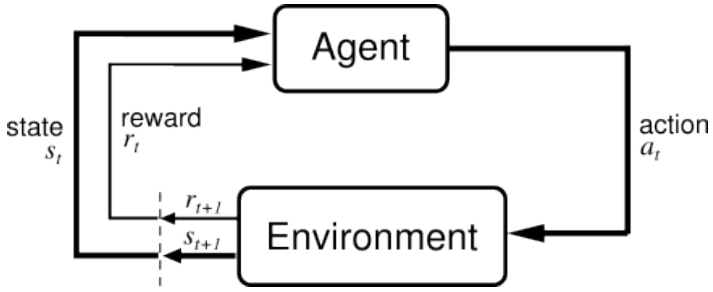


Figure 2.1. Reinforcement Learning cycle

2.1.1 Exploitation & Exploration

The concept of exploration is linked to human nature. For instance, we tend to eat in a restaurant where we experienced a good dinner rather than in a restaurant we don't know. In this way, however, we will never know if there is a better restaurant. Exploitation is always doing the same action that always gets us the best value from a state (it is often called greedy action). In exploration, the agent performs different actions in different states to try to learn all the possibilities available and thus obtain a better reward in the long run. The trade-off between exploitation and exploration is one of the greatest challenges of Reinforcement Learning, to allow the Agent to explore the environment but at the same time to exploit what she has learned and repeat the path found that maximizes the reward.

2.1.2 On-Policy & Off-Policy

The Agent chooses the actions to be performed in each state based on a policy, but this does not happen when the algorithm is in the learning phase. On-policy algorithms seek to improve or evaluate the policy used to make decisions. On the contrary, unlike off-policy algorithms that evaluate or improve a policy other than the one used to generate the data. Two famous learning algorithms are Q-Learning and SARSA, the first is an "On-Policy" algorithm, and the second "Off-Policy".

2.1.3 Planning & Learning

RL problems can be model-based or model-free. In a Model-based RL problem, the agent exploring the environment tries to understand the world in which he finds himself and then creates a model that represents it. Here the model tries to capture two functions, the state transition function T and the reward function R to plan optimal actions. Using this approach we are modelling our environment. In Model-free RL, the agent has no

prior knowledge and uses the “trial-and-error” technique to obtain information. Using this approach there is no need to have information about the environment.

2.1.4 Markov Decision Process (MDP)

Markov Decision Process is a type of mathematics model widely used in machine learning, in this case in RL. The model determines the ideal behaviour within a specific environment. MDP is defined by the state, model, action, reward, and finally by a policy that tries to solve the problem. MDP is based on the property of Markov which states: "The future is independent of the past given the present." When we know the current state, all the past information is no longer needed, the current state is sufficient as if all history were contained in it. In mathematical terms:

$$\mathcal{P}[St+1 \mid St] = P[St+1 \mid S1, \ldots., St] \tag{2.1}$$

The state captures all relevant information from history. The transition function from a Markov state S to its successor state is defined by:

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right] \tag{2.2}$$

It is the probability distribution that the agent given the current state has of going to the next possible successor states.

A Markov process, also known as Markov Chain is a tuple (S, P) on state space S, and transition function P and is a sequence of random states also called memoryless random process.

Markov reward process

A Markov Reward Process or MRP is a tuple (S, P, R, γ) where S represents finite states, P is the state transition probability function, R is a reward function. R is defined as follows:

$$\boldsymbol{R}s = \mathbb{E}[\boldsymbol{R}t+1 \mid \boldsymbol{S}t = S] \tag{2.3}$$

Indicate at this time how much immediate reward we expect to get.

The γ discount factor greatly influences the MDP, it informs the agent how much to worry about future rewards. The γ can vary between 0 and 1 ($\gamma \in [0, 1]$), the higher the discount factor the longer the visual horizon of the agent. If $\gamma < 1$ the agent will try to

get rewards as quickly as possible we are therefore in the *finite horizon* setting. If $\gamma = 1$ the agent will try to get the rewards regardless of time, as if it were *immortal*, in this case, the setting will be *infinite horizon*. In case $\gamma = 0$ the setting is *greedy*, the agent is short-sighted and only looks at the possible reward he could get by performing a single action.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.4}$$

Where γ is between 0 and 1 ($\gamma \in [0, 1]$) The discounted return is indicated with G_t and is the total of the prizes discounted in a given time step t and $R_{t+1}, R_{t+2}, \ldots, R_T$ is the sequence of reward.

Bellman Equation

In order to get the sum of the cumulative rewards, we can use Bellman's equation. Bellman's equation is defined as follows:

$$v_\pi(S) = E_\pi \left(R_{t+1} + \gamma v_\pi \left(S_{t+1} \right) \mid S_t = S \right) \tag{2.5}$$

If we use Bellman's equation, we can break the value function into two parts; a discounted value of the next state $\gamma V(S_t + 1)$ and an immediate reward $R_t + 1$.

State-value function $v_\rho(s)$:

$$v_\rho(s) \coloneqq \mathbb{E}_\rho[G_t | S_t = s], \forall s \in S \tag{2.6}$$

therefore:

$$\begin{aligned} &= \mathbb{E} \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s \right] \\ &= \mathbb{E} \left[R_{t+1} + \gamma \left(R_{t+2} + \gamma R_{t+3} + \ldots \right) \mid S_t = s \right] \end{aligned} \tag{2.7}$$

Now starting from the time step t + 1, we replace the return Gt +1.

Action-value function for policy ρ:

$$q_\rho(s, a) \coloneqq \mathbb{E}_\rho[G_t | S_t = s, A_t = a], , \forall s \in S, \forall a \in A \tag{2.8}$$

Definition 2.1. An *optimal policy* ρ^* is a policy such that $\rho^* \geq \rho$ for all ρ.

A typical reinforcement learning problem is finding the optimal policy for MDP without knowing the transaction function and the reward. It is possible not to specify

the transition probabilities and the rewards but to access it through a simulator that is restarted many times from a uniformly random state $s_0 \in S$ or a fixed state.

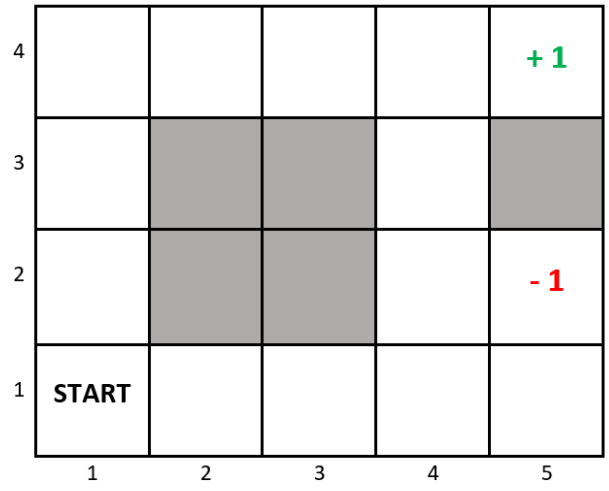
This modelling of the learning process is called *episodic reinforcement learning*. For the MDP simulation to end, the presence of one or more *goal states* is required, in which the activity is considered completed or a maximum time limit for the number of actions that the agent can perform in a single episode, so when the time is greater than the maximum time ($t > T$) the episode ends. It is possible to define a *failure states*, where the episode ends and the activity is considered failed.

Examples

Many dynamic systems can be modeled as Markov Decision Processes.

Example 2.1 (**Gridworld**). A very simple example of MDP is *Gridworld* shown in the figure. The agent can perform four actions: $A = \{Right, Left, Up, Down\}$ and moves into a grid environment made up of 4×5 cells. The agent starts from a fixed initial state $s_0 = s_{11}$, from the current state the agent can move to one of the adjacent free cells. The goal is to reach the state s_{45} considering an episodic task, while s_{35} represents a failure state. The actions can be deterministic or non-deterministic, in the first case, we can get the best action at each state by following the highest estimate, in case the actions are non-deterministic (as happens in the real situation) the agent will not always go to the position he hopes to go to, i.e. the effect of an action is determined by the probabilistic distribution returned by $T(s, a)$.

Figure 2.2. The Gridworld environment



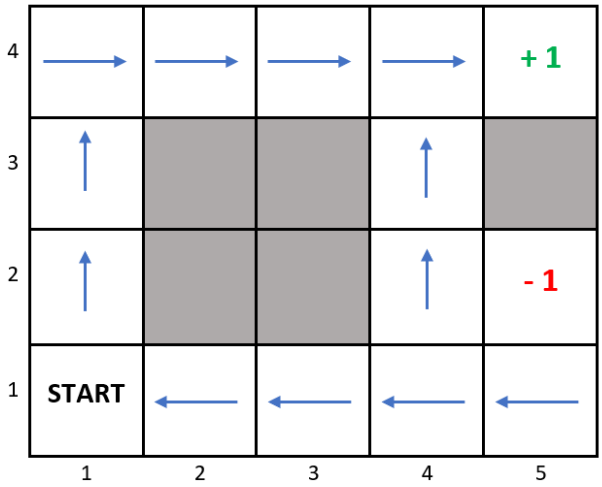


Figure 2.3. An example of optimal policy for the Gridworld environment

If the action the agent chooses causes him to hit a wall (borders of the grid and s_{22}, s_{32}, s_{32}, s_{33}, s_{35}), the agent stays where it was. The reward function $R(s, a, s')$ is defined as 1 if the agent gets to $s' = s_{45}$, like -1 if $s' = s_{35}$ so if it fails, and -0.01 otherwise. The -0.01 reward called *step reward* is widely used to encourage the agent to finish the episode as soon as possible. Since it takes multiple steps to reach the goal, the γ discount factor should be greater than 0.

After several episodes, you will get an optimal policy such as the one shown in Figure 2.3. The optimal action of the state s_{14}, is not to go up (shortest path) but to go left. This is to prevent the agent from ending up in the s_{25} state of error, even if the probability is very low. So the agent prefers to take a lot of small negative rewards, rather than getting a large negative reward equivalent to failure.

2.1.5 RL 'Learning' algorithm

The Reinforcement Learning algorithms can be divided into 3 categories: Value-Based, Policy-Based, Actor-Critic; it is also possible to divide the different models according to on-policy or off-policy.

Q-Learning

The Q-learning algorithm is one of the best known Reinforcement Learning algorithms. It is part of the "Value-Based" category of algorithms and is off-policy. Let's imagine the scenario in which a drone starting from a point on the map must cross the whole map to

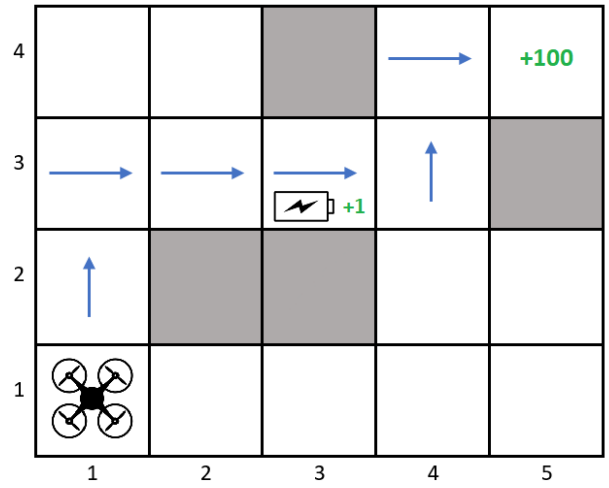


Figure 2.4. Optimal policy for the Gridworld environment

reach its goal. The allowed movements of the drone are: $A = \{Right, Left, Up, Down\}$, if the drone hits an obstacle, the agent is given a negative reward of 100 points and the game fails. If the drone passes over a charging station it gains power, so it earns 1 point. In case the drone reaches the final goal it gets 100 points. The goal is to reach the final goal in the shortest possible time, without hitting an obstacle, to solve this problem the first step is to create a Q-Table. The Q-Table is a table that follows the form of $[state, action]$, our Q-table is made up of 5 rows and 4 columns and we have 4 actions for each state (5x4x4). Initially, the Q-table has all values initialized to 0, after a certain amount of episodes, it ultimately finds the best state-action pairs $Q(s, a)$ for the algorithm, the so-called "Q-values". This q-table becomes a reference table for our agent to select the best action based on the q-value. The Agent interacts with the environment in two ways: "exploitation", "explore", we have already defined what it means in chapter 2.1.1.

Our drone initially knows nothing of the environment, so it will choose actions randomly, as the UAV explores the environment the epsilon rate decreases and the drone begins to exploit the environment. When the Q-tables is ready, the agent will begin to take the best actions, as we can see in Figure 2.4 the drone begins its path by moving upwards s_{21} and not to the left s_{12} which is an equivalent path, this is because along the path taken there is a charging station s_{33} where he will earn an additional reward.

Here's how our Q-table is updated at each time step t:

$$Q(s_t, a_t) = \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \times \left[\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right] \tag{2.9}$$

where:

- $\alpha \in [0, 1]$ is the *learning rate*, a coefficient that regulates how much the newly learned values will contribute in the update
- $\gamma \in [0, 1]$ is the *discount factor*, a coefficient that controls the weight of future rewards. Values closer to 0 will make our agent "short-sighted", considering only the immediate rewards.

Then the algorithm is the following:

Algorithm 2.1 Watkin's Q(λ) (Watkins, 1989)

Require:

S is a set of states
A is a set of actions
γ the discount reward factor
α is the learning rate
n is number of episodes to run Q-learning
ϵ, probability to take random action, rather than follow policy
procedure Q-LEARNING
Initialize $Q(s, a)$ will all 0 utility values.
    for each episode e_i with $i = 0...n$ **do**
     Initialize s
        for each step of episode **do**
            Choose a_t from s_t using policy derived from Q with ϵ-Greedy
            Take action a_t, observe reward r and s_{t+1}
            Update Q-table using equation 2.9
        end for
    end for
end procedure

After a fixed number of steps or after a final state each episode converges. The action choice in Step 12 uses the exploration/exploitation policy (function) $EEP(s_t, Q_t, S, A)$ that is defined as follows:

$$EEP(s_t, Q_t, S, A) = a_t = \begin{cases} \underset{a \in A}{arg\max}\ Q_t(s_t, a_t) & \textit{exploitation} \text{ with probability } 1 - \epsilon \\ \\ \underset{a \in A}{random}(a) & \textit{exploration} \text{ with probability } \epsilon \end{cases}$$

The above EEP function implements a policy denoted as the "greedy policy" where ϵ is often chosen as a small probability (i.e., 0.05).

SARSA Algorithm

SARSA like Q-Learning is part of the *Value Based* algorithm category, but unlike Q-Learning it is an *on-policy* algorithm. The name SARSA derives from its modus operandi, the agent passes from a pair of *state-action* values to another pair of *state-action* values and along the way collects the reward R (so its the *S (t), A (t), R (t + 1), S (t + 1)* & *A (t + 1)* tuple that creates the term S, A, R, S, A). The main difference between SARSA and Q-learning (Watkins and Dayan, 1992) is that Q-learning uses the Q over all possible actions for the next step; while SARSA uses Q by executing an ϵ-greedy policy, as action A is selected by it.

Algorithm 2.2 Sarsa(λ) (Singh and Sutton, 1996)

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$ for all s, a
for each episode **do**
    initialize s
    Choose a from s using policy derived from Q (e.g. e-greedy)
    for each iteration step **do**
        Take action a, observe reward r and new state s'
        Choose a' from s' using policy derived from Q
        $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
        $e(s, a) \leftarrow 1$ ▷ replacing traces
        for all s, a **do**
            $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
            $e(s, a) \leftarrow \gamma\lambda e(s, a)$
        end for
        $s \leftarrow s'$, $a \leftarrow a'$
    end for state s is terminal
end for

2.2 LTL$_f$ and LDL$_f$

In this section, we will describe what LTL$_f$/LDL$_f$ and LDL$_f$ formulas are, what they are for, and the fields in which they are used. Understanding the utility of these formulas is very important because we will use them to define the goal of our RL environment.

2.2.1 Linear-Time Temporal Logic (LTL)

A Linear-Time Temporal Logic formula (LTL) (Pnueli, 1977), is an extension of modal logic and allows to express of temporal patterns on some p properties. It is the most famous temporal logic and is often used in various fields, such as AI, in the verification of software and hardware systems, by companies to verify their processes and specify them declaratively.

Very often LTL formulas are used to express properties and constraints on finite traces of actions/states, this can be done even if the standard semantics of LTL are defined on infinite traces.

Syntax

An LTL formula is defined by a finite set of P symbols and are closed under the boolean connectives, the unary temporal operator, the unary temporal operator $\bigcirc$(*next-time*) and the binary operator $\mathcal{U}$ (*until*):

$$\varphi \quad ::= \quad A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

Boolean abbreviations such as: $\vee, \Rightarrow, \Leftrightarrow$, *true*, *false* and temporal formulas like *eventually* as $\Diamond\varphi \doteq true\mathcal{U}\varphi$, *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$ and *release* as $\varphi_1 \mathcal{R} \varphi_2 \doteq \neg(\neg\varphi_1 \mathcal{U} \neg\varphi_2)$.

Example 2.2. Several interesting temporal properties can be defined in LTL:

- *Liveness*: $\Diamond\varphi$, which means "condition expressed by φ *at some time* in the future will be satisfied", "sooner or later φ will hold" or "eventually φ will hold". E.g., $\Diamond famous$ (eventually I will become famous), $Request \implies \Diamond Response$ (if someone requested the service, sooner or later he will receive a response).

- *Safety*: $\Box\varphi$, which means "condition expressed by φ, *every time* in the future will be satisfied", "always φ will hold". E.g., $\Box happy$ (I'm always happy), $\Box\neg(temperature > 25)$ (the temperature of the room must never be over 25).

- *Response*: $\Box\Diamond\varphi$ which means "at any instant of time there exists a moment later where φ holds". This temporal pattern is known in computer science as *fairness*.

- *Persistence*: $\Diamond\Box\varphi$, which stand for "There exists a moment in the future such that from then on φ always holds". E.g. $\Diamond\Box detachedapple$ (at some point the apple will detach from the tree and will be detached forever.)

- *Strong fairness*: $\Box\Diamond\varphi_1 \implies \Box\Diamond\varphi_2$, "if something is attempted/requested infinitely often, then it will be successful/allocated infinitely often". E.g., $\Box\Diamond ready \implies \Box\Diamond run$ (if a process is in ready state infinitely often, then infinitely often it will be selected by the scheduler).

2.2.2 Linear Temporal Logic on Finite Traces (LTL$_f$)

The LTL$_f$ formulas are based on a finite sequence of instants (De Giacomo and Vardi, 2013), so on finite tracks instead of infinite tracks as in LTL. However, this small difference has a great impact on the interpretation of the meaning of the formulas. The syntax is the same as the LTL formulas (Pnueli, 1977) and also the LTL$_f$ formulas are defined by a

finite set of symbols P and are closed under the boolean connectives, the unary temporal operator, the unary temporal operator $\bigcirc$(*next-time*) and the binary operator $\mathcal{U}$ (*until*):

$$\varphi \quad ::= \quad A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

A: atomic propositions
$\neg\varphi,\ \varphi_1 \wedge \varphi_2$: boolean connectives
$\bigcirc\varphi$ "next step exists and at next step (of the trace) φ holds"
$\varphi_1 \mathcal{U} \varphi_2$:"eventually φ_2 holds, and φ_1 holds until φ_2 does"

With $A \in \mathcal{P}$.

$$\begin{aligned} \varphi_1 \vee \varphi_2 &\doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ \varphi_1 \Rightarrow \varphi_2 &\doteq \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \Leftrightarrow \varphi_2 &\doteq \varphi_1 \Rightarrow \varphi_2 \wedge \varphi_2 \Rightarrow \varphi_1 \\ \mathit{true} &\doteq \neg\varphi \vee \varphi \\ \mathit{false} &\doteq \neg\varphi \wedge \varphi \end{aligned}$$

And for temporal formulas:

$$\varphi_1 \mathcal{R} \varphi_2 \doteq \neg(\neg\varphi_1 \mathcal{U} \neg\varphi_2) \tag{2.10}$$

$$\Diamond\varphi \doteq \mathit{true} \mathcal{U} \varphi \tag{2.11}$$

$$\Box\varphi \doteq \neg\Diamond\neg\varphi \tag{2.12}$$

$$\bullet\varphi \doteq \neg\bigcirc\neg\varphi \tag{2.13}$$

$$\mathit{Last} \doteq \bullet\mathit{false} \tag{2.14}$$

$$\mathit{End} \doteq \Box\mathit{false} \tag{2.15}$$

Formula: (2.11) $\Diamond\varphi \doteq \mathit{true} \mathcal{U} \varphi$: "φ will eventually hold"
Formula: (2.12) $\Box\varphi \doteq \neg\Diamond\neg\varphi$ "from current till last instant φ will always hold"
Formula: (2.13) $\bullet\varphi \doteq \neg\bigcirc\neg\varphi$ "if next step exists then at next step φ holds" (weak next)
Formula: (2.14) $\mathit{Last} \doteq \bullet\mathit{false}$ denotes last instant of trace.

Semantics

A finite track is a finite word on the alphabet $2^{\mathcal{P}}$. Notice that, a trace π can be seen as a word on a path of a Kripke structure. (Clarke et al. (1999)) We use the following notation. We denote the *length* of a trace π as $length(\pi)$. We denote the i_{th} *position* on the trace as $\pi(i) = L(s_i)$, i.e. the propositions that hold in the i_{th} state of the path, with $0 \leq i \leq last$ where $last = length(\pi) - 1$ is the last element of the trace. We denote by $\pi(i, j)$, the *segment* obtained from π, starting from position i and terminating in position $\pi(j)$, with $0 \leq i \leq j \leq last$

Definition 2.2. Given an LT_f-interpretation π, we define that a LTL$_f$ formula φ is *true* at time i $(0 \leq i \leq last)$, in symbols $\pi, i \models \varphi$ as follows:

$$\pi, i \models A, \text{ for } A \in \mathcal{P} \text{ iff } A \in \pi(i)$$

$$\pi, i \models \neg\varphi \text{ iff } \pi, i \not\models \varphi$$

$$\pi, i \models \varphi_1 \wedge \varphi_2 \text{ iff } \pi, i \models \varphi_1 \wedge \pi, i \models \varphi_2$$

$$\pi, i \models \bigcirc\varphi \text{ iff } i < last \wedge \pi, i+1 \models \varphi \tag{2.16}$$

$$\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2 \text{ iff } \exists j.(i \leq j \leq last) \wedge \pi, j \models \varphi \wedge$$

$$\forall k.(i \leq k < j) \Rightarrow \pi, k \models \varphi_1 \tag{2.17}$$

Theorem 2.1 (De Giacomo and Vardi (2013))**.** *Satisfiability, validity and entailment for* LTL$_f$ *formulas are* PSPACE*-complete.*

Theorem 2.2 (De Giacomo and Vardi (2013); Gabbay et al. (1997))**.** LTL$_f$ *has the same expressive power of* FOL *over finite ordered sequences.*

As we can see from the two previous theorems the LTL$_f$ formulas have the same expressive power as FOL and have a satisfiability for PSPACE-hardness.

2.2.3 Linear Dynamic Logic on Finite Traces (LDL$_f$)

LDL$_f$ (De Giacomo and Vardi, 2013) has the same decorativeness as LTL$_f$ which is merged with the expressiveness of RE$_f$ but using the syntax of PDL (Fischer and Ladner, 1979), a (propositional) logic often used in computer programs. RE$_f$ is much more expressive than LTL$_f$ but it is a very low-level formal consideration for temporal logics, this is because there is no direct construct for negation and conjunction.

Syntax

Formally, LDL$_f$ formulas φ are built over a set of propositional symbols $\mathcal{P}$ as follows (Brafman et al., 2017):

$$\begin{aligned} \varphi &::= tt \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\varrho\rangle\varphi \\ \varrho &::= \phi \mid \varphi? \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \varrho^* \end{aligned}$$

- tt and *false* stand for true and false
- ϕ: propositional formula on current state/instant
- $\neg\varphi,\ \varphi_1 \wedge \varphi_2$: boolean connectives
- ϱ is a regular expression on propositional formulas
- $\langle\varrho\rangle\varphi$: exists an "execution" of RE$_f$ ϱ that ends with φ holding
- $[\varrho]\varphi$: all "executions" of RE$_f$ (along the trace!) end with φ holding

For classical logical operators we use these abbreviations:

$$\begin{aligned} \varphi_1 \vee \varphi_2 &\doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\ \varphi_1 \Rightarrow \varphi_2 &\doteq \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \Leftrightarrow \varphi_2 &\doteq \varphi_1 \Rightarrow \varphi_2 \wedge \varphi_2 \Rightarrow \varphi_1 \\ ff &\doteq \neg tt \end{aligned}$$

And for temporal formulas:

$$[\varrho]\varphi \doteq \neg\langle\varrho\rangle\neg\varphi \tag{2.18}$$

$$End \doteq [true]ff \tag{2.19}$$

$$Last \doteq \langle true\rangle End \tag{2.20}$$

(2.18) $[\varrho]\varphi$ and $\langle\varrho\rangle\varphi$ are correspond to the box and diamond analogues in LTL.
(2.19) It means that the trace is finished.
(2.20) Denotes the last element of the track.

Semantics

Let us now formally see the semantics for LDL$_f$.

Definition 2.3. Given a finite trace π, we define that a LDL$_f$ formula φ is *true* at time i ($0 \leq i \leq last$), in symbols $\pi, i \models \varphi$ inductively as follows:

$\pi, i \models tt$

$\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$

$\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1 \wedge \pi, i \models \varphi_2$

$\pi, i \models \langle\phi\rangle\varphi$ iff $i < last \wedge \pi(i) \models \phi \wedge \pi, i+1 \models \varphi$

$\pi, i \models \langle\psi?\rangle\varphi$ iff $\pi, i \models \psi \wedge \pi, i \models \varphi$

$\pi, i \models \langle\varrho_1 + \varrho_2\rangle\varphi$ iff $\pi, i \models \langle\varrho_1\rangle\varphi \vee \langle\varrho_2\rangle\varphi$

$\pi, i \models \langle\varrho_1; \varrho_2\rangle\varphi$ iff $\pi, i \models \langle\varrho_1\rangle\langle\varrho_2\rangle\varphi$

$\pi, i \models \langle\varrho^*\rangle\varphi$ iff $\pi, i \models \varphi \vee i < last \wedge \pi, i \models \langle\varrho\rangle\langle\varrho^*\rangle\varphi$ and ϱ is not *test-only*

Now let's compare the LTL$_f$ formulas with the corresponding LTL$_f$ formulas:

LTL$_f$	LDL$_f$
A	$\langle A\rangle tt$
$\neg\varphi$	$\neg\varphi$
$\varphi_1 \wedge \varphi_2$	$\varphi_1 \wedge \varphi_2$
$\bigcirc\varphi$	$\langle true\rangle(\varphi \wedge \neg End)$
$\varphi_1 \mathcal{U} \varphi$	$\langle(\varphi_1?; true)^*\rangle(\varphi_2 \wedge \neg End)$

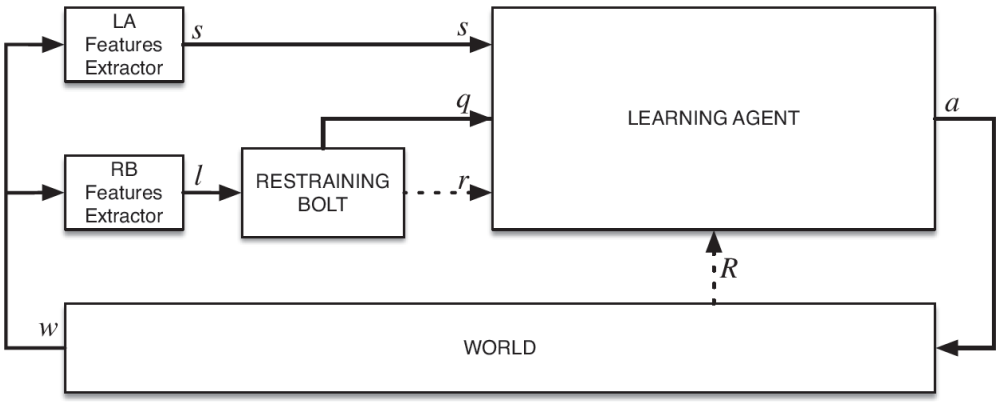
Theorem 2.3 (De Giacomo and Vardi (2013))**.** LDL$_f$ *has exactly the same expressive power of* MSO

2.3 Reinforcement Learning with LTL$_f$ and LDL$_f$

In the previous paragraphs, we have understood what Reinforcement Learning is and what LDL$_f$ and LTL$_f$ formulas are. In this section, we will understand how we can use both of them to solve different types of problems. In our thesis project in particular in our environment in addition to the agent, there is another fundamental component that allows us to manage priorities through the LDL$_f$ and LTL$_f$ formulas, this is called Restraining Bolt (De Giacomo et al., 2019). The Restraining Bolt as in the sci-fi movie

Star-wars is something that limits the actions of the droid, similarly, our RB will limit the actions of our agent. The agent learns in the environment about a set of low-level (sub-symbolic) characteristics to achieve its goals, but while this happens the RB limits its actions to high-level restrictive specifications established by the LTL_f and LDL_f formulas. The goal is for the agent to comply as closely as possible with the restrictive specifications of the RB. The world model is a *Markov decision-making process (MDP)*, hidden and factored on a certain set of (sub-symbolic) world characteristics. The agent state is made up of observable characteristics, while the transition function and the reward function are hidden. There are therefore two distinct sets of characteristics, one set for the agent and another for the RB, which are not related but are related by the world itself (Brooks, 1991). To make RB work well with a Reinforcement Learning agent there is no need to

Figure 2.5. Learning Agent and Restraining Bolt



formalize these correlations. If everything happens correctly the agent's objectives will be met and the restrictions given by the $\text{LTL}_f/\text{LDL}_f$ formulas will not be violated.

The LTL_f and LDL_f formulas can be transformed into deterministic finite state automata (De Giacomo and Vardi, 2013). This allows to transform an NMRDP problem with non-Markovian $\text{LTL}_f/\text{LDL}_f$ rewards into an MDP, with an extended state space, obtained as a product of the states of the automaton with the states of the NMRDP.

2.3.1 RL with Restraining Bolts

- We have: An agent and a MDP model $\mathcal{M}_{ag} = \langle S, A, T, R, \gamma \rangle$ with Tr_{ag} and R_{ag} hidden but sampled from the environment.

- A restraining bolt $\text{RB} = \langle \mathbf{L}, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$ where:

- $\mathrm{L} = 2^{\mathcal{F}}$ L is the set of characteristics of fluents. Analogous to S which denotes the set of features available for M_{ag}.The fluents F do not continue to form the S states of the agent M_{ag}.
- $\{(\varphi_i, r_i)\}_{i=1}^{m}$ is a set of restraining specifications with $*\varphi$, an $\mathrm{LTL}_f/\mathrm{LDL}_f$ formula over F. $*r_i$, the reward associated with φ_i. A reward r_i is assigned to sequences of configurations $\ell_1, \cdots, \ell_n$ satisfying φ_i.

From a formula φ we get a corresponding formula DFA $\mathcal{A}_{\varphi_i} = \left\langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i \right\rangle$. The agent does not see the configurations of the fluents and due to the non-Markov nature the agent does not see the rewards coming from the RB. We therefore provide the agent with further observable characteristics $Q_1 \times \ldots \times Q_m$, which correspond to the satisfaction states of the formulas $\varphi_1 \ldots \varphi_m$. A solution to the problem M_{ag}^{rb} is a (possibly non-Markovian) policy $\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \rightarrow A$ that maximizes the expected cumulative reward.

1. For every φ_i, compute the equivalent DFA $\mathcal{A}_{\varphi_i}$ (De Giacomo and Vardi, 2013)
2. Do RL on the MDP$M_{ag}^{q} = \langle Q_1 \times \cdots \times Q_m \times S, A, Tr'_{ag}, R'_{ag} \rangle$ where:
   - The transition distributionTr'_{ag} is unknown;
   - The rewardR'_{ag}, unknown to the agent, is defined as:

$$R'_{ag}\left(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s'\right) = \sum_{i : q'_i \in F_i} r_i + R_{ag}\left(s, a, s'\right) \tag{2.21}$$

   - The states q_i of the DFAs $\mathcal{A}_{\varphi_i}$ are progressed correctly by the environment.

This is a possible solution for Markov's policies of the form $(Q_1 \times \ldots \times Q_m \times S) \rightarrow A$. It is possible to find another possible solution with the following theorem:

Theorem 1. *RL* with $\mathrm{LTL}_f/\mathrm{LDL}_f$ restraining specifications $M_{aq}^{rb} = \langle M_{ag}, RB \rangle$ with $M_{ag} = \langle S, A, Tr_{ag}, R_{ag} \rangle$ and $RB = \langle L, \{(\varphi_i, r_i)\}_{i=1}^{m} \rangle$ can be reduced to *RL* over the MDP $M_{ag}^{q} = \left\langle Q_1 \times \cdots \times Q_m \times S, A, Tr'_{ag}, R'_{ag} \right\rangle$ and optimal policies $\rho_{\mathrm{a\ g}}^{\mathrm{new}}$ for $M_{\mathrm{a\ g}}^{rb}$ can be learned by learning corresponding optimal policies for M_{ag}^{q}.
Proof. see (De Giacomo et al., 2020).

2.4 BUBBLES Project

The scenario of this thesis project was inspired in some way by the BUBBLES project. BUBBLES is a project funded by the European community, coordinated by public and private bodies. The BUBBLES consortium includes three academic partners:

- Universitat Politècnica de València (UPV): UPV is the coordinator of the project, deals with the separation of ConOps and OSED / SPR / INTEROP, and is coordinated by Professor Juan Vicente Balbastr.
- Universidade de Coimbra (UC): UC deals with the communication-related aspects of ConOps separation management, coordinated by Professor Henrique Madeira.
- University of La Sapienza in Rome (UNIROMA1): The DIAG is responsible for applying Artificial Intelligence methods to the separation management ConOps and is coordinated by Professor Luca Iocchi.

The other two partners are:

- EUROCONTROL: it is participating in the ConOps definition and validation by simulation, as well as in performing Performance and Safety Operational Assessments.
- Indra Sistemas S.A. (Indra): Indra is responsible for defining scenarios supporting the ConOps definition and participates in the concept of experimental validation.

2.4.1 BUBBLES objectives

The BUBBLES project aims to define U-space services, this will allow the operation of a large number of drones without interfering in any way with manned aviation. In manned aviation there is a very low accident rate (less than 3 per million departures), this high safety is due to the CNS concept, first introduced by ICAO for PBN in 1998. However, U-space services for UAS are still being defined, especially those (U3). According to the SESAR organization, more than 450,000 UAS will be expected to fly in the airspace VLL (Very Low Level) SES (below 150 m), for this reason, it is important to improve safety and efficiency by introducing advanced services. The BUBBLE project will manage this concept of UAS separation management in U space through artificial intelligence (AI) based algorithms to calculate the collision risk between UAS leading to minimal separation and methods to maintain a target level of security (TLS). These algorithms will be applied to a series of generic ConOps and will be classified in terms of risk using the SORA (Specific Operations Risk Assessment) methodology. SORA is a risk assessment process of some unmanned aircraft operations, divided into several assessment phases. In the BUBBLES project, the concepts of collision and conflict are used, two concepts that may seem similar but are different. Conflict is when the distance between two or more UAVs can general in the future a potential collision, by collision we mean

the physical impact of a drone against some static physical structure or a moving object. Such concepts will often be used in defining issues and scenarios in this thesis project.

Chapter 3

Background

3.1 OpenAI Gym

OpenAI Gym is an open-source toolkit for the development and comparison of Reinforcement Learning algorithms, Gym is based on Python and is compatible with all numerical computation libraries such as *TensorFlow* and *Theano*.

Gym was publicly released in 2016 by the non-profit organization OpenAI, founded by *Elon Musk* and *Sam Altman* in 2015 based in San Francisco.

Gym's goal is to offer a benchmark for reinforcement learning that is easy to use and implement, with a large variety of different but similar environments. It is possible to use either predefined environments for known problems such as *"MountainCar"* or it is possible to create a new environment from scratch, as has been done for our framework. Another goal of the *OpenaAI Gym* library is to standardize how environments are defined in scientific articles, a bit like it happens for *ImageNet, CIFAR* that make available large quantities of labelled data to make progress in the field of scientific research.

Let's now see an example of implementation of the famous Gym environment *MountainCar*. First, we will name the environment, we call it *Car_v0*. Five important Python commands are common in all Gym environments.

- *gym.make*(*EnvironmentName*) creates the environment, you can choose known environments present on the site and on the OpenAI Gym GitHub repository or you can create a new environment from scratch.

- *env.reset*(): this command is used to reset the environment and initialize it at the first observation.

- The *for t in range (5000)*: loop executes an instance of the *Car_v0* environment

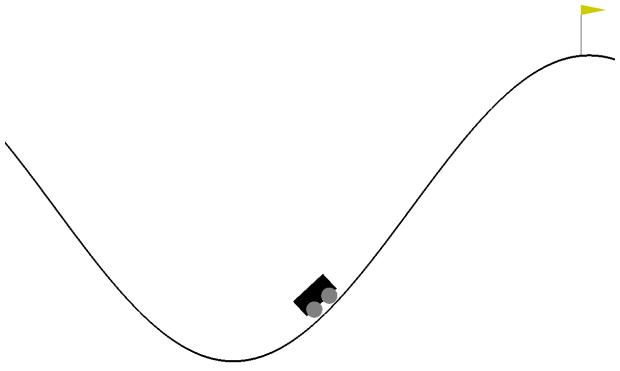
for *5000* iterations, rendering at each pass,

- *env.render*(): in this way, being inside the loop, after each iteration a pop-up window will open which will graphically reproduce the problem.
- *env.step*(): in our code this command will perform an action at each step. In general *env.step()* returns four parameters which are: *observation, reward, done and information*.

Listing 3.1. `MountainCar`

```
import gym
env = gym.make('Cart_v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(5000):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode␣finished␣after␣{}␣timesteps".format(t+1))
            break
env.close()
```

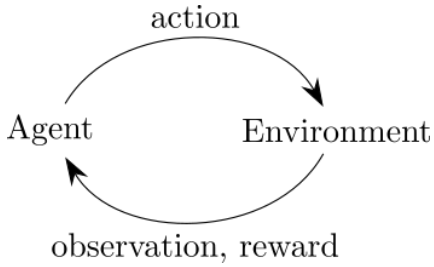
Figure 3.1. MountainCar Environment



Let's now understand what these four values are returned by the "step" command.

- *observation*(*object*): The observation is a specific object that represents the observation of the environment, it could be, for example, the speed and the angle of curvature of a car, the pixel data of a camera, or the state of the board of a board game.
- $reward(float)$: The *reward* is the reward obtained from the previous action carried out by the agent, it can be positive or negative and the goal is to maximize this reward.
- *done*(*boolean*): If done is set to true, it allows the episode to end. For example, two *UAVs* collide, so done is True and the episode ends.
- $info$: Sometimes it can contain raw probabilities after the last change of state of the environment, it can be useful for debugging. However, the agent cannot use any of this information.

Figure 3.2. Agent-Environment loop



At each time step, the agent performs an action and then observation and a reward are returned.

As we see in the picture, our car must reach the top of the hill. We have three discrete actions: *push left: 0, no push: 1, push right: 2*. The observations are instead of the *box* type since the environment is discrete and are position and speed. The reward is *-1* for each time phase until the target position is reached. To reach the target position the car must take a run, then push left and right until the car has enough momentum to reach the flag.

3.2 MultiUAV-OpenAIGym and gym-sapientino

In addition to the EU SESAR BUBBLES project that inspired my thesis scenario, two other projects influenced the development of this thesis work. Some components of these

works will be used, such as the restraining bolt, to specify $\text{LTL}_f/\text{LDL}_f$ time formulas. These two works are MultiUAV-OpenAIGym and gym-sapientino.

3.2.1 MultiUAV-OpenAIGym

It is an environment written in python based on the OpenAi-Gym framework, which is extended to obtain a multipurpose environment for UAVs. In MultiUAV-OpenAIGym (Brunori, 2020) policies are generated through Reinforcement Learning for autonomous multi-drone systems in multi-service applications. The available RL algorithms are SARSA and Q-Learning, with which it is possible to find policies for each agent. The UAVs present in the environment provide network or mobile computing services to users. The goal is to make UAVs learn, through reinforcement learning, how to provide these services, maximizing the quality of experience (QoE).

Adopted notation

The notations adopted in MultiUAV-OpenAIGym are the following:

Parameters	Notation
Number of UAVs	N
Number of Charging Stations	M
Number of Users	U
Number of Users Clusters	C
Number of Users in the i-th Cluster	U_{C_i}
Number of Covered Users	U_{C_i}
Needed Battery for the i-th UAV to get to the elosest CS	B_i
j-th Reward Structure for i-th UAV	BW_i
Remaining Bandwidth of the i-th UAV	critical $_i$
i-th Critical Battery level	needed $_{B_i}$
Maximum extension along Y axis for the planar xy-grid map	$\mathcal{R}_j^i$
Maximum extension along X axis for planar xy-grid map	L
Maximum extension along Z axis for 3D map	W

Table 3.1. Used notation for the for the environment parameters.

Framework settings

There are several settings supported by this framework, such as choosing the desired resolution for the cells, deciding whether to consider an environment with coordinates (x, y) then 2D or 3D (x, y, z), how many drones to fly and at what time interval t must start.

It is possible to randomly generate buildings by setting a percentage of obstacles for the entire operating volume, it is also possible to choose a maximum height of the obstacle/building, in this way obstacles with different heights will be generated.

It is possible general of the centroids in which to distribute users, using a normal distribution. The service requested by users can be defined as (throughput, edge computing or data collection).

Charging stations (CS) can be inserted into the environment, equidistant from each other, where the drones can go to recharge to continue providing their services.

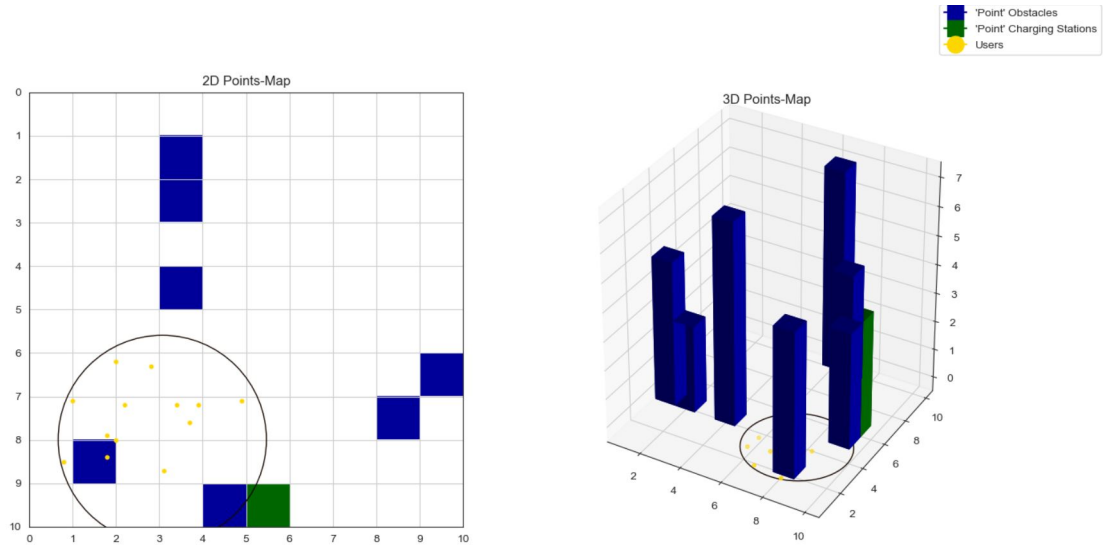


Figure 3.3. Multi-Service_UAV: on the left is represented the 2D grid-map view, while on the left is shown the 3D view

Reward Functions

Now let's see in detail the three reward functions implemented. As the constraints of the problem and the training parameters vary, the reward function changes.

$$\mathcal{R}_1^i = \frac{U_c}{\hat{U}}, \text{ where } \hat{U} = \frac{U}{N} \tag{3.1}$$

$$\mathcal{R}_2^i = w_s r_u + w_c r_c \tag{3.2}$$

where:

$$w_{s,c} = \begin{cases} \begin{cases} 1.0 \\ 0.0 \end{cases}_{c}^{s} & \text{if } B_i > critical_1 \\ \begin{cases} 0.8 \\ 0.2 \end{cases}_{c}^{s} & \text{if } (critical_1 < B_i \leq critical_1) \wedge (B_i > needed_{B_i}) \\ \begin{cases} 0.5 \\ 0.5 \end{cases}_{c}^{s} & \text{if } (critical_3 < B_i \leq critical_2) \wedge (B_i > needed_{B_i}) \\ \begin{cases} 0.2 \\ 0.8 \end{cases}_{c}^{s} & \text{if } (critical_4 < B_i \leq critical_3) \wedge (B_i > needed_{B_i}) \\ \begin{cases} 0.0 \\ 1.0 \end{cases}_{c}^{s} & \text{otherwise, i.e. if } B_i \leq needed_{B_i} \end{cases} \tag{3.3}$$

with r_u computed as in (3.1) and the reward cost equals to:

$$r_c = \frac{nedeed_{B_i}}{B_i} \tag{3.4}$$

A third reward function is computed as:

$$\mathcal{R}_3^i = w_s \left(w_u r_u + w_{tr} s_{tr} + w_{ec} s_{ec} + w_{dg} s_{dg}\right) + w_c (r_{cp} + r_{cs}) \tag{3.5}$$

w_u, w_{tr}, w_{ec}, w_{dg} are the weights corresponding respectively to the percentage of:

- covered users (regardless of the service requested);
- covered throughput requests;
- covered edge computing requests;
- covered data gathering requests.

3.2.2 Gym-sapientino

Gym-sapientino (Favorito, 2018) is a framework developed in python, like MultiUAV-OpenAIGym it is also based on the OpenAI Gym framework. As the name suggests, the framework is inspired by the Sapientino educational game, for children aged (5-8). In this environment, the real game is simulated, where there is a mobile robot that has to visit specific cells. Cells contain concepts that must be matched by the children (e.g., a coloured animal, a colour, and the initial letter of the name of the animal). The robot has to move and visit a sequence of cells in a 5x7 grid, when the robot reaches the cell it has to visit, it signals it with a beep.

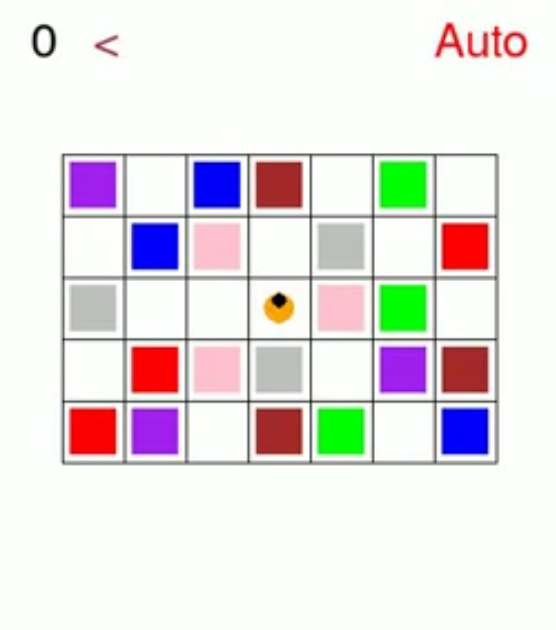


Figure 3.4. A screenshot from gym-sapientino

The agent can perform several actions which are: $(up, down, left, right)$ and BIP. The characteristics for the agent space are the agent's (x, y) position in the grid, that is, f_x and f_y. The configurations of the fluids are evaluated through features, which are: f_b which reports whether a BIP has just been executed, and f_c which denotes the colour of the current cell. In the default settings, the available fluents are:

$$\mathcal{P} = \{beep, red, green, blue, pink, brown, gray, purple\}$$

just map features to fluents one-to-one.

Framework settings

In gym-sapientino the algorithm used to find the agent's policy is SARSA (λ) (e.g. Sarsa with eligibility traces) (Singh and Sutton, 1996) with ϵ-greedy policy. The training parameters are set as follows:

- $\lambda = 0$
- $\gamma = 1.0$
- $\alpha = 0.1$
- $\epsilon = 0.1$

Temporal Goal

The time formula set by default and used for the experiment is a LDL_f formula:

$$\begin{aligned}&\langle true^*; red \wedge bip; true^*; green \wedge bip;\\ &\ true^*; blue \wedge bip; true^*; pink \wedge bip;\\ &\ true^*; brown \wedge bip; true^*; grey \wedge bip;\\ &\ true^*; purple \wedge bip\rangle tt\end{aligned} \tag{3.6}$$

Visit each colour and do a *bip* in each colour in a given order

To perform this experiment, (5000) episodes were performed. The illegal beep was punished with a negative reward. In this way the agent recognizes, earlier than no reward shaping configuration , that the illegal *bip* is a bad action.

Restraining bolt

In this environment was introduced the concept of the Restraining bolt which manages the time specifications. Several experiments have been made with different time formulas in this case we see a LTL formula implemented.

$$\begin{aligned}&\neg \,\text{bip}\,\mathcal{U}\,(\text{V}_{j=1,2,3}\,\text{cell}_{C_1,j} \wedge \text{ bip }) \wedge\\ &\textstyle\bigwedge_{j=1,2,3} \Box\,(\text{cell}_{C_1,j} \wedge bip \rightarrow \bigcirc\Box\,(\text{bip} \rightarrow \neg\,\text{cell}_{C_1,j})) \wedge\\ &\textstyle\bigvee_{j=1,2,3} \Box\left(\text{cell}_{C_1,j} \wedge \text{bip} \rightarrow \bigcirc\left(\neg\,\text{bip}\,\mathcal{U}\bigvee_{k\neq j}\text{cell}_{C_1,k} \wedge \text{ bip }\right)\right.\end{aligned}$$

This LTL formula specifies that the agent must visit at least two cells of the same colour for each colour in a given order (the colour order is predefined: first $C1$, then $C2$, and so on).

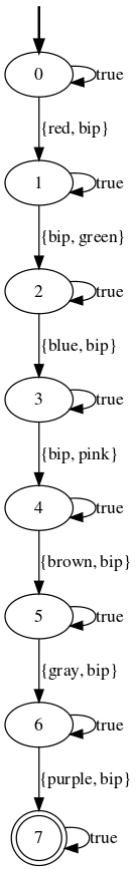


Figure 3.5. The automaton associated to the LDL_f formula in Equation 3.6

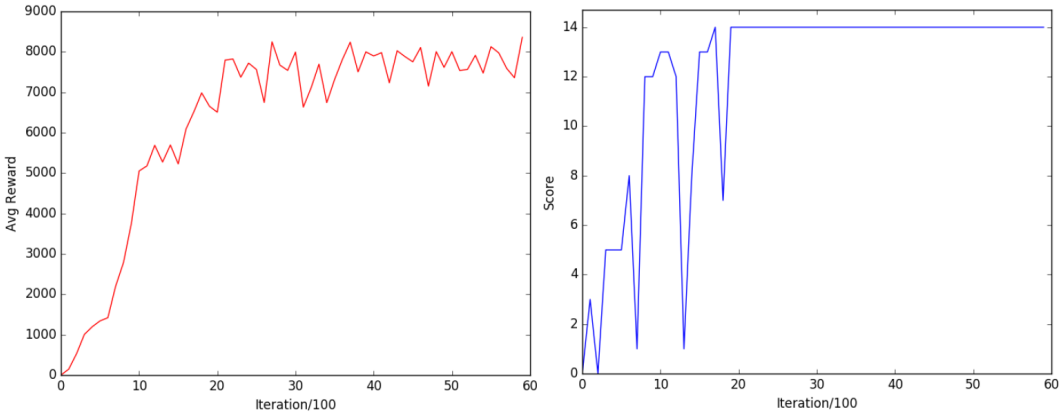


Figure 3.6. Sapientino S2 OMNI (3 minutes)

Chapter 4

Problems Definition

In this chapter, we provide an abstract definition of the problem and then focus on an instance of the problem that led us to the definition of the scenario of the thesis. We consider a set of reinforcement learning agents, where each agent has its own goal and they all operate in a shared environment. The goal of this thesis work is to find policies that are both optimal for each agent to achieve their goal and ensure a very low probability of conflict, thus minimizing the possibility of interference between the policies of one or more agents.

More formally, the considered problem can be described in terms of the following elements.

- A world W (for example an environment, a room, a video game). Let W be the set of *world states*, i.e. the states of the world W.
- $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$ is the set of agents, each agent α_i has a set of possible actions A_i and a set of states S_i. $\delta_i : S_i \times A_i \to S_i$ (or a probability distribution $P(s' \mid s, a)$) and an initial state $s_i^0 \in S_i$
- Each agent α_i has a goal M_i, with $\mathcal{M} = \{M_1, \ldots, M_m\}$ being the set of tasks and $f : \mathcal{A} \to \mathcal{M}$ a function assigning one task to each agent.
- Each agent α_i generates a trajectory T_i representing the sequence of states crossed during the execution of the optimal policy.
- The goal for each agent α_i is to find an optimal policy that will produce a trajectory τ_i, ensuring a minimal risk of conflict.

- The frequency of the conflict for two trajectories are obtained by multiplying the frequency of the encounters in two points by the probability that these two points are within the conflict threshold, i.e.

$$F_{\text{conflict}}\ (\tau_i, \tau_j) = \min(f_i, f_j) \Pr(|x_i - x_j| < \delta)$$

where τ_i and τ_j are two trajectories of two agents α_i, α_j, with $i \neq j$. f_i and f_j are the frequency of the trajectory ($f_i = \frac{1}{T_i}$). On the trajectories t_i and t_j we have respectively the points $x_i \in \tau_i$ and $x_j \in \tau_j$ which represent two agents.

The frequency of the event in which the two agents are in a given position x_i and x_j is determined by the minimum frequency, i.e. $min(f_i, f_j)$, while δ indicates the conflict threshold that is the distance within which a conflict is detected.

4.1 Assumptions and Scenario Overview

In this thesis, we will focus on an instance of this abstract problem to define our scenario.

- A Scenario $\mathcal{S} = \langle \mathcal{A}, \mathcal{M}, \mathcal{RB} \rangle$ is a tuple containing a set of agents $\mathcal{A}$, agents modeled by MDP $\mathcal{M}$, a set of $\mathcal{RB}$ (Restraining Bolts) that will define the behaviours of the agents $\mathcal{A}$.
- $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$ is a set of UAV agents, each agent α_i is modeled by the MDP model $\mathcal{M}_{ag_i} = \langle S_i, A_i, Tr_{ag_i}, R_{ag_i}, \gamma \rangle$ with Tr_{ag_i} and R_{ag_i} hidden but sampled from the environment.
  - We assume a global clock, which measures the time during the evolution of the scenario for each agent α_i, from 0 to T.

    Let's consider a set of asynchronous goals, in which each agent can decide to execute its task at any time independently from the other agents. Each agent does not know where the other agents are at any given instant of time. The agent can interact with $\mathcal{W}$ by executing actions A_i and cannot directly communicate or perceive other agents.
  - Let's assume the learning agent has a special action *stop* which deems the end of an episode. The agent can do action a, at every clock and observe both the new state s from the new world state w', namely $s = \mathbf{f}_{ag}(w)$, and a real-valued reward $R(s, a, s')$, that depends only from the transition $s \rightarrow_a s'$.

  - We also assume that the state transition function (Tr_{ag_i}) is Markovian, so the next state s' depends only on the current state s and the action taken a.

- Each agent α_i has a goal (φ_i, P_i), where φ_i is the formula LTL$_f$/LDL$_f$ to be satisfied and P_i is a non-negative integer denoting the priority associated with the goal described by φ_i. $P_i \in \mathbb{Z}^{0+}$, in which the value 0 represents the highest priority. The assignment of agents α_i to goals φ_i with priorities P_i forms a list $\mathcal{AP}$ of agents sorted by goal priorities (from highest to lowest).

  In general, to provide a high-level description of the world we can consider a LTL$_f$/LDL$_f$ formula φ_i $(i = 1, \ldots, m)$ over a set of fluents $\mathcal{F}$. Fluents in F are not among the features that form the states S of the learning agent α_i. We denote by $\mathcal{L} = 2^{\mathcal{F}}$ the set of possible fluent configurations.

- The behaviour of each UAV is learned through reinforcement learning with Restraining Bolts. Each agent α_i has a RB_i, which gives positive or negative rewards whether the LTL$_f$/LDL$_f$ formula is met or not.

  More formally, $\mathcal{RB} = \{RB_1, \ldots, RB_m\}$ is the set of Restraining Bolts augmented with a priority value P_i. Each agent α_i is associated to $RB_i = \langle\, \mathcal{L}, (\varphi_i, r_i, P_i)\rangle$, where $\mathcal{L}$ is the set of possible fluent configurations shared among all agents, φ_i is a LTL$_f$/LDL$_f$ formula over $\mathcal{L}$, r_i is the reward associated with φ_i, and P_i is the priority of the goal described by this RB. During the learning process, the agent receives the rewards from both RB_i and from R_{ag_i}.

  Note that the number of agents α_i may differ from the number of Restraining Bolts RB_i $(n \neq m)$. There is no one-to-one association between agents α_i and Restraining Bolt RB_i , but it is possible to have scenarios where one or more RB_i are not assigned to any agent α_i or scenarios where the same RB_i is duplicated on one or more agents α_i. For example, if two agents have to perform the same mission, both agents will have two copies of the same RB, while if an RB is not associated with an agent, it means that no agent is carrying out the mission associated with that RB.

  While in the original article (De Giacomo et al., 2019), RB goals were formally defined with a set of formulas, in this thesis we will consider cases in which only one formula φ_i is used to denote the goal within an RB.

  A possible solution to the problem is a (possibly non-Markovian) policy $\bar{\rho} : (Q_1 \times \ldots \times Q_m \times S)^* \rightarrow A$ where the cumulative reward is maximized.

4.2 Solution concept

Our solution is given by n policies ρ_{ag_i}, one for each agent α_i, that are individually optimal concerning each agent's goal, and that in the case of conflicts, they are always resolved in favour of the agent who has a goal with higher priority (lower value of P_i).

To eliminate interference between agent policies, we define a new "no-interference" reward function N_{ag_i} for each agent. The N_{ag_i} function is computed by applying, to the original reward function R_{ag_i} a modifier that depends on the trajectories generated by agents with higher priority goals.

Each agent must satisfy its φ_i formula and must respect the priority value assigned to this goal.

- If the agent α_i has $P_i = 0$ (highest priority), then N_{ag_i} equals the reward function R_{ag_i}.
- If the agent α_i has $P_i > 0$, then N_{ag_i} reward function is the sum of the reward function R_{ag_i} of the agent α_i and of a modifier that depends on the trajectories generated by optimal policies of agents α_j for which $P_j < P_i$.

More formally

$$N_{ag_i} = R_{ag_i} + f(\{\tau_j | \alpha_j \ s.t. \ P_j < P_i\}) \tag{4.1}$$

N_{ag_i} is defined on the basis of the policies obtained from the agents α_j, who have a higher priority $P_j < P_i$.

Note that, we admit non-deterministic behaviour during optimal policy execution. However, we assume that the variability of the trajectories is limited so that it is possible to define a modifier function f removing interference.

4.2.1 Formal solution

By using the no-interference reword functions N_{ag_i}, previously defined instead of R_{ag_i}, we eliminate the interference among the m MDPs making them all independent of each other. In this case, we can apply the method described in (De Giacomo et al., 2019) incrementally and derive an optimal multi-agent behaviour, where multi-agent optimality is defined in terms of the modifier function f.

Let's solve the problem more formally:

We have a tuple $\mathcal{S} = \langle \mathcal{A}, \mathcal{M}, \mathcal{RB} \rangle$, with $\mathcal{M}_{ag_i} = \langle S_i, A_i, Tr_{ag_i}, N_{ag_i}, \gamma \rangle$ and $RB_i = \langle \mathcal{L}, (\varphi_i, r_i, P_i) \rangle$.

- For each φ_i, compute the equivalent DFA $\mathcal{A}\varphi_i$ is (De Giacomo and Vardi, 2013).
- Apply RL on our MDP $M^q_{ag_i} = \left\langle Q_{1_i} \times \cdots \times Q_{m_i} \times S_i, A_i, Tr'_{ag_i}, N'_{ag_i} \right\rangle$
  - The Tr'_{ag_i} is not known
  - The reward N'_{ag_i}, is unknown to the agent α_i, and is defined as follows:

$$N'_{ag_i}\left(q_{1_i}, \ldots, q_{m_i}, s_i, a_i, q'_{1_i}, \ldots, q'_{m_i}, s'_i\right) = \sum_{j:q'_j \in F_j} r_{j_i} + N_{ag_i}\left(s_i, a_i, s'_i\right)$$

  - The environment correctly progresses the q_i states of the DFA $\mathcal{A}\varphi_i$

By exploiting the theorem in (De Giacomo et al., 2019), our scenario $\mathcal{S} = \langle \mathcal{A}, \mathcal{M}, \mathcal{RB} \rangle$, can be reduced to RL over extended MDPs and optimal policies $\rho^{new}_{ag_i}$ for $M^{rb}_{ag_i}$ can be learned by learning corresponding optimal policies for $M^q_{ag_i}$.

From the eecution of the optimal policy $\rho^{new}_{ag_i}$, we get a trajectory τ_i that will be used in the next iteration to calculate new no-interference reward N_{ag_j} for agents with lower priority goals ($P_j > P_i$).

4.3 Algorithm

Now let's define an algorithm we need to get the solution to our problem.

Algorithm 4.1 No-interference

$\mathcal{A}$ is a set of agents
$\mathcal{M}_{ag_i} = \langle S_i, A_i, Tr_{ag_i}, R_{ag_i}, \gamma \rangle$ each agent α_i is modeled by the MDP model
$RB_i = \langle \mathcal{L}, (\varphi_i, r_i, P_i) \rangle$ is the Restraining Bolt
α_i is the agent
R_{ag_i} is the reward functions associated with the agent α_i
$\mathcal{L}$ is the set of possible fluent configurations shared among all agents
φ_i is the formula $\text{LTL}_f/\text{LDL}_f$ to be satisfied by the agent α_i
r_i is the reward associated with φ_i
$\mathcal{P}_i$ is a non-negative integers and is the priority associated with the agent α_i
$\mathcal{AP}$ is the list of agents α_i sorted by goal priorities $(P_0, \dots P_{n-1})$
for α_i in $\mathcal{AP}$ **do**:
    $N_{ag_i} \leftarrow R_{ag_i} + f(\{\tau_j | \alpha_j \ s.t. \ P_j < P_i\})$
    $\rho_{ag_i} \leftarrow$ RL solution of $\mathcal{M}_{ag_i} = \langle S_i, A_i, Tr_{ag_i}, N_{ag_i}, \gamma \rangle$ (De Giacomo et al., 2019)
    $\tau_i \leftarrow$ execution of policy ρ_{ag_i}
end for
return set of policies $\{\rho_{ag_i}\}$

The No-interference reward function N_{ag_i} is the sum between the reward function associated with φ_i and of a modifier that depends on the trajectories generated by optimal policies of agents α_j.

4.4 Theoretical analysis

The function f is a domain-dependent function, which therefore cannot be explained in this abstract and theoretical chapter. Our hypothesis in which f manages to isolate and thus to determine independence from the rewards is true, then the solution we get is the one we have previously described.

In the next section, we will show how our implementation of f will make the solution of our case study optimal.

The algorithm defined in the previous section starts by iterating each α_i agent according to its priority. The agent with the highest priority $P = 0$ is always the first to perform its task and the no-interference reward function N_{ag_i} is always equal to R_{ag_i}.

Since α_0 is the first agent to run, there is no τ_j interference. The agent with the highest priority $P = 0$ is always the first to perform its task and the no-interference reward function N_{ag_i} is always equal to R_{ag_i}.

Observing the algorithm in the previous section we can note that the reward function M_{ag_i} is replaced by a no-interference reward N_{ag_i}.

$$\mathcal{M}_{ag_i} = \langle S_i, A_i, Tr_{ag_i}, N_{ag_i}, \gamma \rangle$$

$$RB_i^+ = \langle \mathcal{L}, (\varphi_i, r_i, P_i) \rangle$$

We eliminate the interferences between the m MDP making them all independent from each other, it is possible to apply the algorithm of (De Giacomo et al., 2019) to solve the problem and obtain the optimal policy ρ_{ag_i} for each agent α_i. By executing the optimal policy ρ_{ag_i}, we obtain the τ_i trajectory that we will use in the next iteration to calculate our new no-interference reward N_{ag_i}.

Note that the no-interference reward function N_{ag_i} for each agent α_i, which is equal to the sum of the reward function R_{ag_i} with τ_j of the agent α_j, τ_i is generated from execution of the optimal policy ρ_{ag_i} returned from the algorithm of (De Giacomo et al., 2019).

4.5 Case study

Our case study is an instance of the previously defined abstract problem. Let's assume we have a global clock that measures the time during the evolution of the scenario for each agent α_i, from 0 to T. Let's consider a set of asynchronous goals, each agent does not know where the other agents are at any given instant of time.

The scenario of our case study is a tuple defined as follows: A Scenario $\mathcal{S} = \langle \mathcal{A}, \mathcal{M}, \mathcal{RB} \rangle$. Where $\mathcal{A}$ is the set of α_i modeled by the MDP, in our case each agent is a UAV. $\mathcal{RB}$ is the set of Restraining Bolts, which define the behavior of the UAV in order to reach the goal. The agent interacts with the W world through actions A_i.

- Each UAV_i has a (φ_i, P_i) goal. Where φ_i is a LTL$_f$/LDL$_f$ formula that specifies the order in which each UAV_i must visit hospitals in order to complete the mission. P_i is a non-negative integer associated with the φ_i goal that indicates the priority of each UAV_i. The UAV_i with the highest priority has priority $P_i = 0$, the one with the lowest priority has priority $P_i = n - 1$.

- There are five actions allowed for each agent α_i. $A_i \in [0, 4]$, where actions 0 to 3 correspond to a direction (up, down, left or right) and action number 4 corresponds to the "Hovering / beep" state.
- In our case, the RB_i of an agent α_i gives a positive reward if the agent performs the "Hovering/beep" action first on the cell from which the agent must take the resource and then on the cell where the UAV_i must release the resource.
- To find our no-interference reword we apply equation 4.1. In our case f has been implemented by adding negative weights on the cells crossed by the trajectories τ_j of the previous agents α_j. In this way, the UAV_i when it passes over a cell in which the UAV_j has already passed receives through the no-interference reward function N_{ag_i} a negative reward. This leads the agent UAV_i to find an optimal trajectory to reach his goal, considering the trajectories τ_j performed before him by the UAV_j having higher priority.

4.5.1 Reward values

In our case, each agent α_i generates a trajectory that is not necessarily unique (because the policy, in general, can be non-deterministic), which corresponds to the set of states crossed during the execution of the optimal policy. For each agent, ρ_{ag_i} is a set of cells (x, y) that α_i visited before reaching its goal.

The reward function computes a reward for each step t depending on the action done and the state in which the agent is:

$$r_t(s_t, a_t) = \begin{cases} 10 & \text{if } s_t = \varphi_i \quad \text{(agent reached the goal)} \\ -1 & \text{if } s_t = badBeep_i \quad \text{(the agent makes a "beep"/"hovering")} \\ -1 & \text{if } s_t \in N_{ag_i} \quad \text{(agent moved over previous paths)} \\ -1/(width \cdot height) & \text{off map} \quad \text{(agent moved)} \\ -0.1 & \text{if } s_t \neq \varphi_i \quad \text{(agent has not yet reached the goal)} \end{cases}$$

Chapter 5

Implementation

In this chapter, we will describe the implementation of this thesis work. In particular, the possible settings and the most important parts of the code.

As specified in the previous chapters 4.4, the problem defined above was implemented as a reinforcement learning problem. The code was written in python and the OpenAI Gym library was used to create the environment or it is compatible with other environments sharing a similar structure (same action space and state space) and with other RL algorithms that follow Gym APIs.

The libraries used in addition to the OpenAI Gym toolkit are:

- OpenAI Gym;
- numpy;
- decimal;
- sklearn;
- matplotlib;
- mpl_toolkits;
- scipy;
- statistics;
- ffloat;
- pythomata;

5.1 Environment setting

The code is easily extendable and it is possible to make different types of settings. It is possible to set the number of hospitals in the scenario, the priority of each hospital in the form of colour, the number of UAVs, the priority of each UAV based on the mission it has to carry out, the number of obstacles in case you pass to the 3D environment, whether the UAV has an unlimited battery or not. It is also possible to choose which type of algorithm to use and implement new ones, the algorithms implemented at this time are Q-learning and SARSA, however, the objectives of this thesis go beyond the optimization of RL methods.

5.1.1 Q-Table Initialization

It is possible to initialize the Q-tables in different ways, in our environment it is possible to use four different types of initializations when using Q-learning:

1. zero initialization;
2. random initialization;
3. maximum reward initialization;
4. prior initialization;

- The *zero-initialization* is the most common, it is possible to set all the value of the q-tables to 0, in this way every all the state-action combinations are set to the initial value 0. However, this setting could lead to stalemate. If a random step is not provided, if this stalemate occurs the agent is led to always perform the same sequence of actions.
- Another approach is to initialize the *Randomly inizalization.* In this case, always assuming that no random steps are used, it is possible to avoid the 'local minima' (i.e. a stuck sequence of actions), the training time could increase.
- *Maximum Reward Initialization* surely there will be no local minimums so the UAVs will not get stuck in learning but it could lead to a very high training time. A value much higher than the upper limit set for the reward is associated with each pair of shares.
- *prior initialization* This implementation builds on previous knowledge and is useful for roughly letting the UAVs know the location of the hospitals they need to reach. The assumption is that the map can be divided into six sectors as "good" or "bad".

5.2 Priority management

The management of priorities is an important part of the code, as already mentioned in section 4.5, each goal has an associated priority and it is possible to associate the goal and priority to a specific UAV.

5.2.1 UAVs priority

The UAVs are sorted by priority, the highest priority UAV being the first to iterate. By executing the newly obtained policy, a trajectory is generated and stored in an array. The trajectory is composed of points (x, y) in the 2D case. When the second iteration begins, a no-interference reward is created which contains the reward function of the UAV that is performing its mission plus the negative reward weights associated with each point (x, y) that the UAV0 has passed through. In this way, the UAV1 takes a negative reward every time it passes over the cells inside this array. This procedure occurs at each iteration, so each UAV always tries not to pass over a point (x, y) where another UAV has already passed. With this method we prevent UAVs from interfering with each other, thus ensuring safety.

5.2.2 Restraining Bolt

The Restraining Bolt is another important element for priority management. Here is a piece of implementation:

Listing 5.1. Restraining Bolt.

```
@staticmethod
def extract_UAV_fluents(obs) -> PLInterpretation:
    fluents = set()
    if (UNLIMITED_BATTERY == True):
        beep = obs[0]['beep']
        color_idx = int(obs[0]['color'])
    if ((0 < color_idx <= len(RestrainingBolt.get_colors())) and beep):
        color_string = RestrainingBolt.get_colors()[color_idx - 1]
        fluents.add(color_string)
    elif ((color_idx == 0) and beep):
        fluents.add("bad_beep")
    result = PLInterpretation(fluents)
    return result
```

In the first section of code with the function *extract_UAV_fluents* we get the fluents of the hospitals which we will then need to build our automata. The condition is true in the second "if" when the agent *hovering/beep* on a colour that is not an empty cell. If the colour is 0 (ie the cell has no colour) when the UAV does *hovering/beep*, a "*bad_beep*" is added to the set of fluents.

Listing 5.2. Restraining Bolt.

```
def step_agent(self, agent, action):
    """Do a step in the Gym environment."""
    obs, reward, done, info = self.env.step_agent(agent, action)
    if (UNLIMITED_BATTERY == True):
        color_idx = obs[2]

    action = [action]

    next_automata_states = [tg.step(obs_dict, action)
    for tg in self.temp_goals]

    temp_goal_rewards = [
        tg.observe_reward(is_terminal_state=done)
        for tg in self.temp_goals
    ]
    total_goal_rewards = sum(temp_goal_rewards)

    if any(r != 0.0 for r in temp_goal_rewards):
        logger.debug("No-zero␣goal␣rewards:{}".format(temp_goal_rewards))

    obs_prime = (obs_dict, next_automata_states)
    reward_prime = reward + total_goal_rewards

    return obs_prime, reward_prime, done, info
```

In the second section of code, we can see how the *step_agent* function implemented inside wrapper.py communicates both with the agent and with the Restraining Bolt. The "*env.step_agent*" function extracts the observations, the reward obtained by the agent and the "*done*" at each iteration. The reward accumulated by the agent during the iteration is added to the possible reward given by the Reistrainig Bolt, which in our

case is 10 when the UAV respects the LDL_f formula, so when the UAV *beeps*/*hovering* on the right coloured hospitals order.

Chapter 6

Results

In this chapter, we show and illustrate the empirical results obtained by our framework. The experiments focus on the possible benefits obtained from the use of our algorithm for managing the priorities of 2 or more agents.

6.1 Evaluation Criteria

The Reinforcement Learning algorithm used to carry out the experiments and generate the optimal policies for each agent is SARSA. The results are compared through graphs, which show the reward function behaviour when increasing the number of episodes and the number of steps required in each episode. By carrying out the experiments, we obtain excellent policies that allow us to generate trajectories, which we can view through gifs or images. Viewing the trajectories makes us better understand how our algorithm resolves the conflicts present between the UAVs, showing us how each agent first reaches the hospital that sends the resource and then the recipient hospital.

The total reward plot highlights how many episodes are necessary to complete the objective with the lowest possible penalty. In our case study, reach the hospitals in the shortest possible time respecting the $\text{LTL}_f/\text{LDL}_f$ formulas associated with the missions and avoid the trajectories of the UAV with higher priority. Steps per episode instead can provide a measure of the time required from the agent to reach the goal in the specific step. Lengths plot shows us how many episodes are needed to minimize the actions that each agent takes to complete the goal.

6.2 Experiments

The experiments conducted are three, each consisting of a learning phase and a test phase. Each UAV starts from a different position and at each iteration receives a negative reward of (-0.01) to entice the agent to finish the mission in the shortest possible time.

Parameters	Values
Number of episodes	1000
Number of iterations	50
Number of UAVs	3
Number of Restraining Bolts	3
Number of fluents x Restraining Bolts	2
Number of priority **	3
Discount γ	0.99
Learning Rate α	0.1
Epsilon ϵ	0.1
Unlimited Battery	*True*

** Only by using the no-interference reward, a priority assigned to each agent.

Table 6.1. Environment parameters.

As we can see in table 6.1 we have associated a Restraining Bolt with each agent, each RB has a LDL_f formula. Each LDL_f formula specifies how many fluent our agent must visit before being satisfied, in our case each UAV must visit two hospitals, but we can add more fluents to the formula to specify more goals.

The three LDL_f formulas used to specify the goal φ_i of our agents α_i are:

- **UAV1:** <(!yellow & !orange)*;yellow;(!yellow & !orange)*;orange>tt
- **UAV2:** <(!green & !darkGreen)*;green;(!green & !darkGreen)*;darkGreen>tt
- **UAV3:** <(!blue & !darkBlue)*;blue;(!blue & !darkBlue)*;darkBlue>tt

We assigned a priority to each agent, the UAV with highest priority $P = 0$ is UAV1. UAV2 and UAV3 have $P = 1$ and $P = 2$ respectively.

6.2.1 Case I & II - with and without priority

In image 6.1, case in which the no-interference reward is not applied, we see how the UAV2 (orange colour trajectory) does not respect the priorities to reach the second objective, thus intersecting the trajectory of the UAV1 (purple colour trajectory) with the highest priority. In this case, there is a conflict that must be managed to avoid the collision.

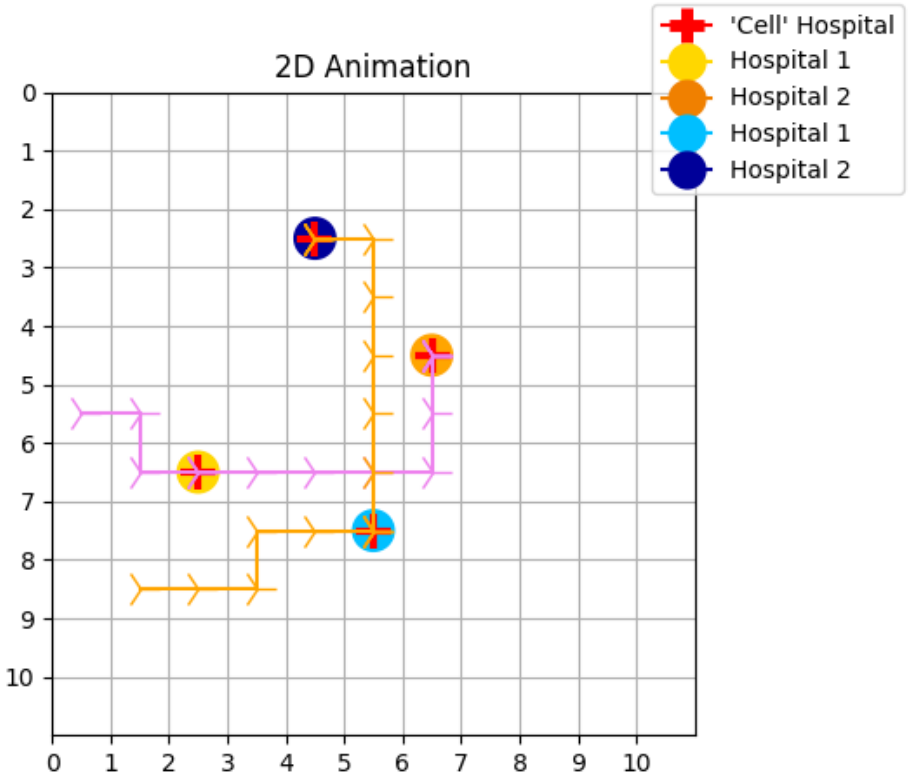


Figure 6.1. Experiment 1. Without priority

In figure 6.2 the UAV1 (purple colour trajectory) with the highest priority reached the first and second hospital in the shortest possible time, making Hoverin / beep on both. The UAV2 (orange colour trajectory) is the second starting as established, can see that the UAV2 reaches its goal without ever intersecting the trajectory of the UAV1, in this case, the risk of collision between the UAVs is 0, have thus maximized safety.

Note that we have assumed that the goals are asynchronous, so each UAV can decide to execute its mission at any time. Each UAV is not aware of the location of the other UAVs at any given instant of time.

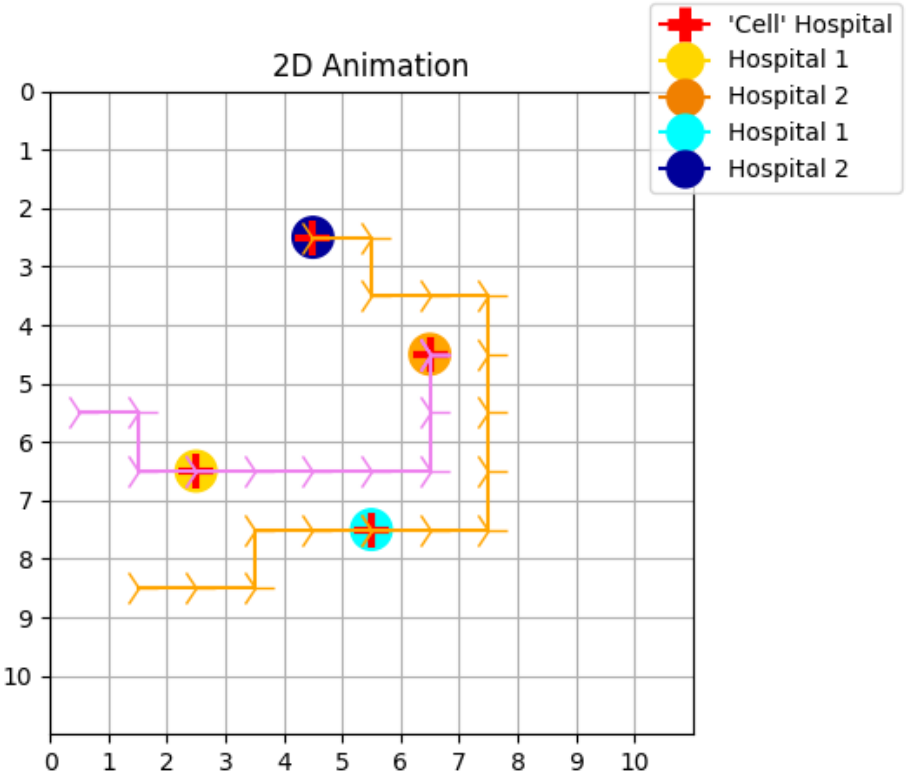


Figure 6.2. Experiment 2. With priority

6.2.2 Case III & IV - with and without priority

In figure 6.3 and 6.4, we can see that we have added two more hospitals and another drone (grey colour trajectory). The latter added UAV has the lowest priority in the scenario. It is easy to see in image 6.2, in this case, have two conflicts. One conflict is between UAV1 and UAV2, the other is between UAV2 and UAV3.

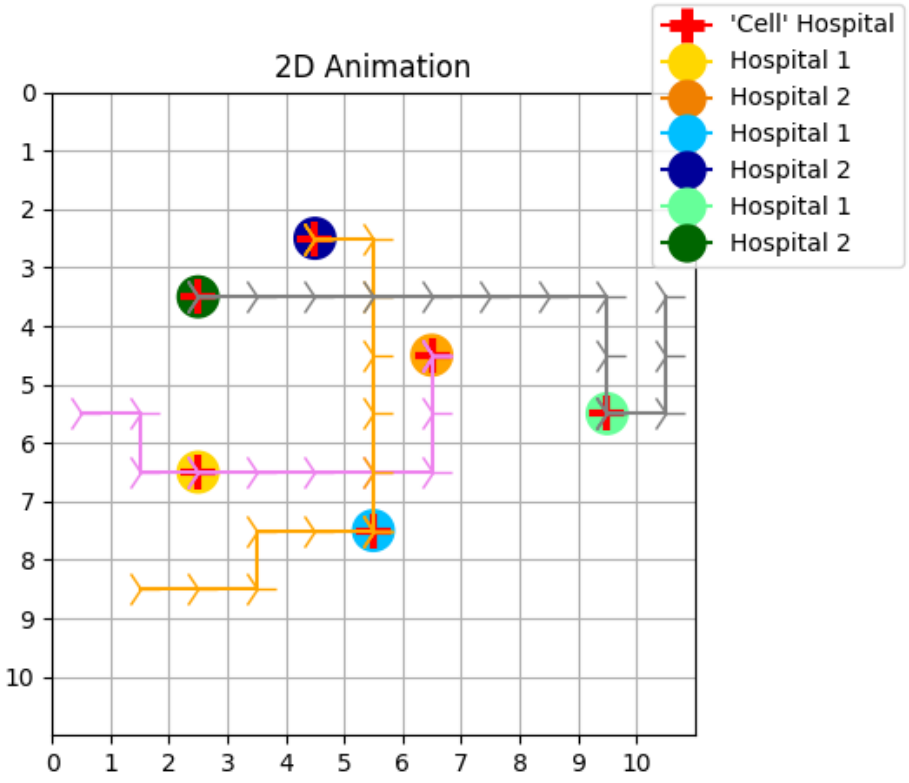


Figure 6.3. Experiment 3. Without priority

In case IV, figure 6.4, we apply the no-interference reward to enforce priorities and resolve conflicts between UAVs. The benefits are obvious, in this case, the highest priority UAV1 reaches its goal first of all in the shortest possible time. UAV2 no longer intersects the trajectory of UAV1, but goes around it, thus resolving the conflict. The UAV3 does not disturb either the trajectory of the UAV1 or that of the UAV2, completing the mission safely. All agents carried out their mission without interference with each other, achieving our main goal, maximizing safety.

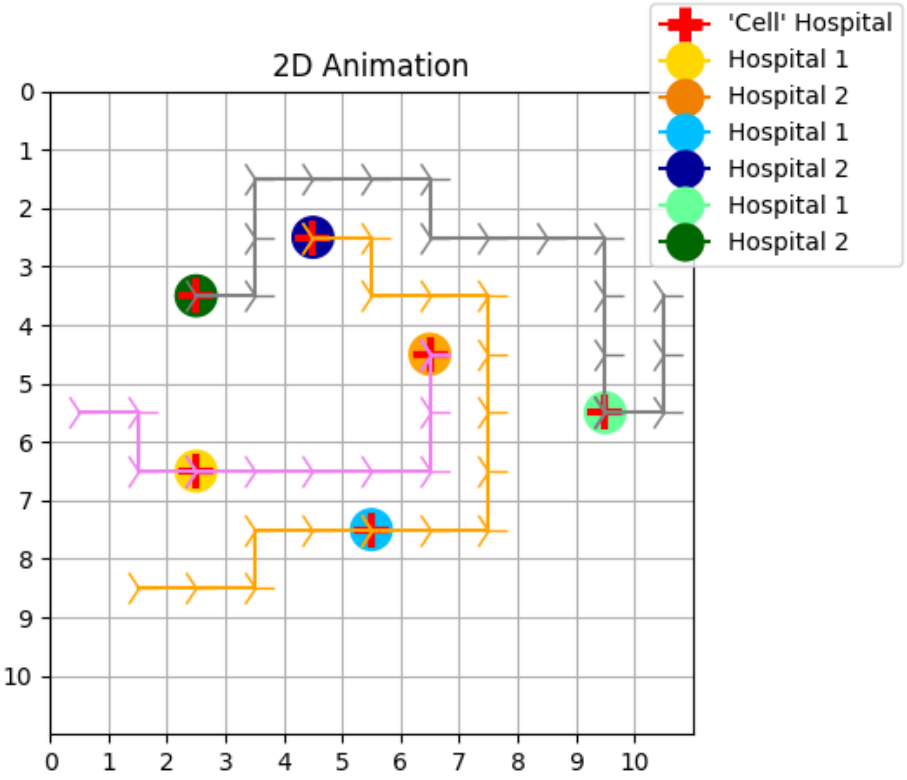
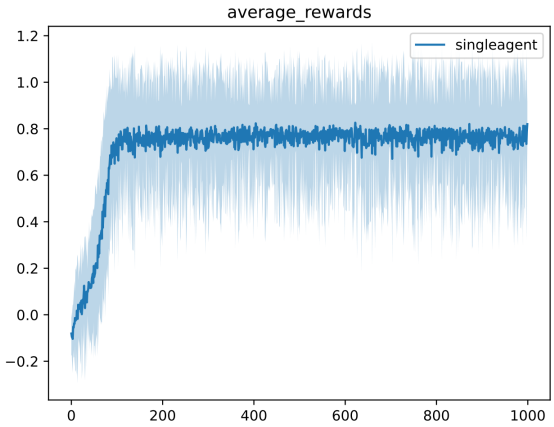


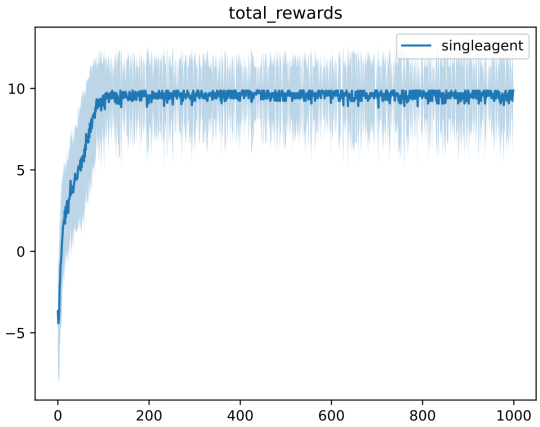
Figure 6.4. Experiment 4. With priority

6.2.3 Case I & II - Performance Comparison

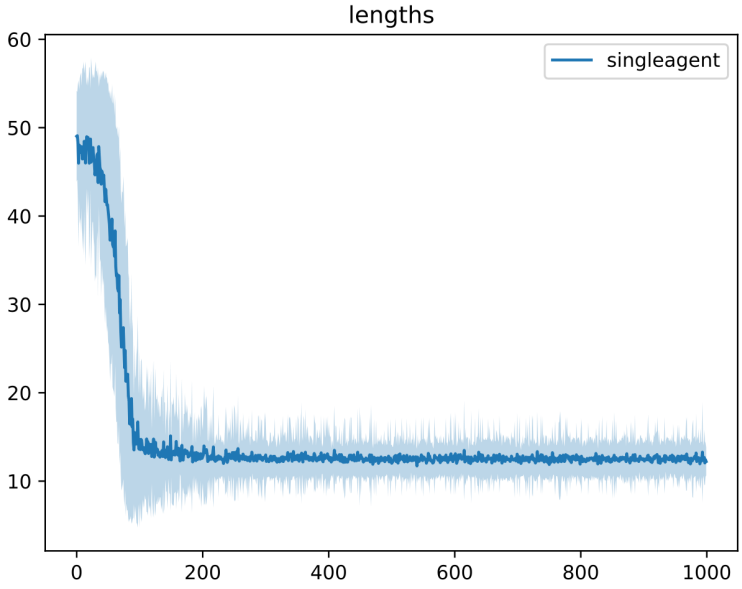
Figure 6.5 includes three graphs regarding the UAV1. These three graphs are the same for both the case with priority and the case without priority UAV1 is the first to start so there are no conflicts.



(a) UAV1 - averange rewards.

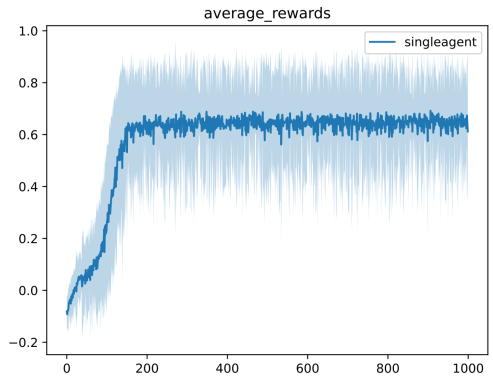


(b) UAV1 - total rewards.

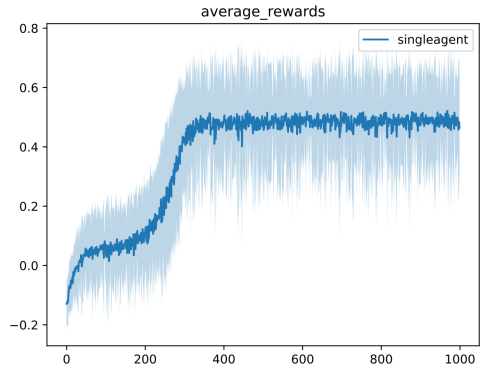


(c) UAV1 - number of steps per episode.

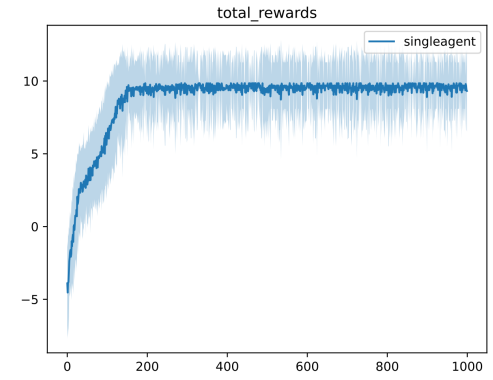
Figure 6.5. CASE I & II.



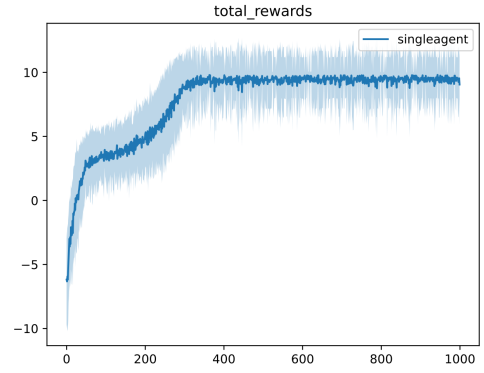
(a) Case I: UAV2 - averange rewards.



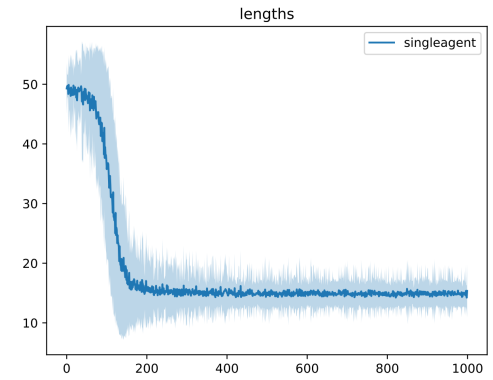
(b) Case II: UAV2 - averange rewards.



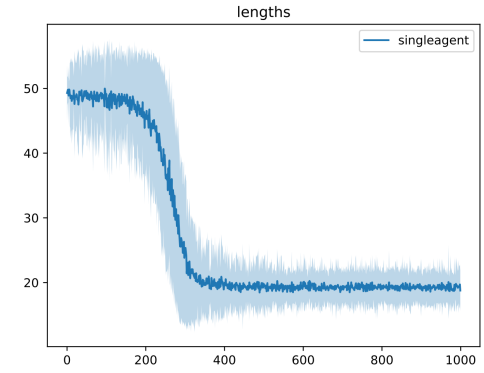
(a) Case I: UAV2 - total rewards.



(b) Case II: UAV2 - total rewards.



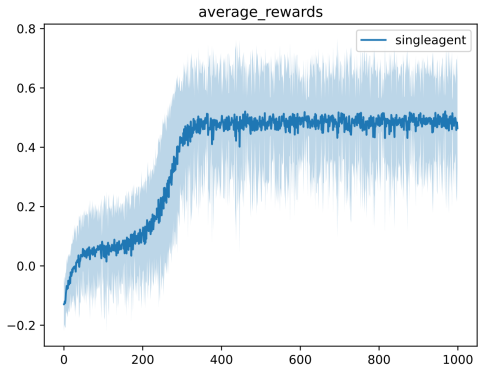
(a) Case I: UAV2 - number of steps per episode.



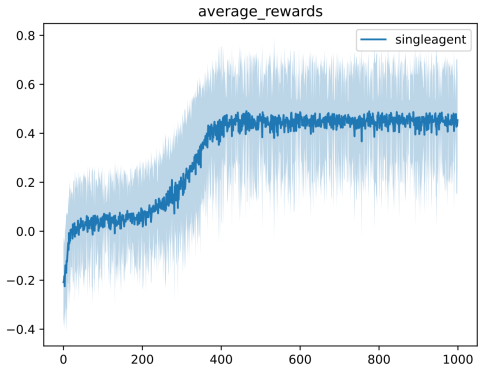
(b) Case II: UAV1 - number of steps per episode.

6.2.4 Case III & IV - Performance Comparison

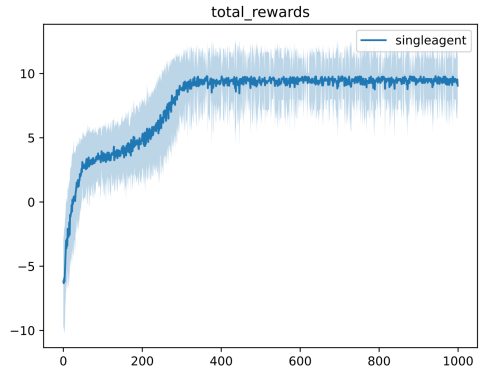
The graphs show us that the UAV3 takes longer to find its optimal policy, as we have already described, to maximize the reward the UAV3 undertakes not to pass over the cells in which the other UAVs have passed.



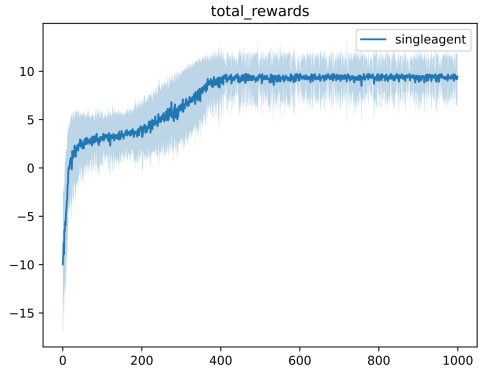
(a) Case III: UAV3 - averange rewards.



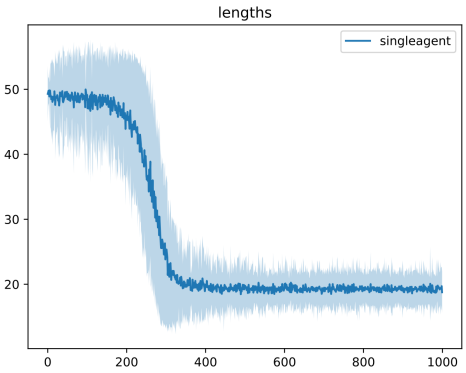
(b) Case IV: UAV3 - averange rewards.

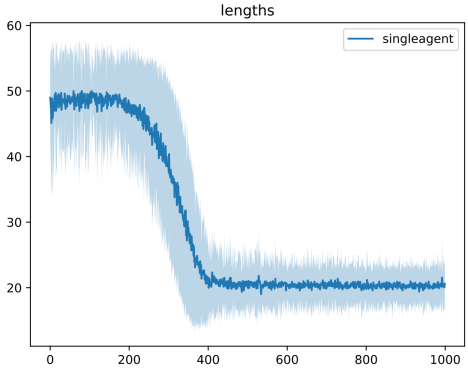


(a) Case III: UAV3 - total rewards.



(b) Case IV: UAV3 - total rewards.





(a) Case III: UAV3 - number of steps per episode. **(b)** Case IV: UAV3 - number of steps per episode.

Chapter 7

Conclusion and Futures Works

This thesis work could be a new way to generate off-line trajectories free of conflicts or in any case with an extremely low potential for conflict. The hospital scenario is just one example of how this system could be used to generate priority-based trajectories. This study allows you to manage two types of priorities. The priorities managed by the Restraining Bolt which, thanks to a $\text{LTL}_f/\text{LDL}_f$ formula, allows to establish in which order the UAV must visit the hospitals, therefore from whom to take the resource and to whom to deliver it. The priorities managed by the no-interference reward function which allows eliminating the disturbances between the policies of each UAV, in this case between the trajectories generated by the execution of optimal policies. In the case study, using the independent learning technique, we demonstrate that we obtain excellent policies that respect the concept of priority. Experiments result empirically demonstrated that all the conflicts present in cases I and III were resolved in cases II and IV, at the expense of a negligible increase in the training time. The no-interference reward function in the recent future could be deepened and submitted to a scientific article. A possible future work is to use the high-level generated trajectories of our work to generate real trajectories. Also, it is possible to consider the case where the goals are not asynchronous, so each UAV could know where another UAV is at a given instant of time.

Bibliography

Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Specifying non-markovian rewards in mdps using ldl on finite traces (preliminary version). *CoRR*, abs/1706.08100, 2017.

R. A. Brooks. Intelligence without representation. *Artif. Intell*, 47:139–159, 1991.

Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-03270-8.

Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, volume 13, pages 854–860, 2013.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. *Proceedings of the International Conference on Automated Planning and Scheduling*, 29: 128–136, Jul. 2019. URL `https://ojs.aaai.org/index.php/ICAPS/article/view/3549`.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Restraining bolts for reinforcement learning agents. *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 13659–13662, Apr. 2020. URL `https://doi.org/10.1609/aaai.v34i09.7114`.

Marco Favorito. Reinforcement learning for ltlf/ldlf goals: Theory and implementation. *Master's thesis. DIAG, Sapienza Univ. Rome*, 2018.

Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194 – 211, 1979. ISSN 0022-0000. doi: https://doi.org/10.1016/0022-0000(79)90046-1. URL `http://www.sciencedirect.com/science/article/pii/0022000079900461`.

D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. Technical report, Jerusalem, Israel, Israel, 1997.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL `http://dl.acm.org/citation.cfm?id=645528.657613`.

Amir Pnueli. The temporal logic of programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society. doi: 10.1109/SFCS.1977.32. URL `https://doi.org/10.1109/SFCS.1977.32`.

Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Mach. Learn.*, 22(1-3):123–158, January 1996. ISSN 0885-6125. doi: 10.1007/BF00114726. URL `http://dx.doi.org/10.1007/BF00114726`.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL `https://doi.org/10.1007/BF00992698`.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, 1989.