# Bachelor Thesis - Drone fleet traffic monitoring

GOUDET Oscar, 314512
PECAUT Marius, 330684
WUNDERLI Léo, 329314

EPFL

Thursday 18th July, 2024

# Contents

# 0 Introduction

Today, urban areas around the world struggle with the significant challenge of traffic congestion. In fact, a minor breakdown within the transportation network can lead to a full congested area with stagnant vehicle flow. Addressing these challenges requires a deep understanding of driving behaviors and commuters habits. Traditionally, this understanding is obtained thanks to extensive infrastructure investments, which are not financially interseting and often fall short in terms of operational efficiency.

Conventional traffic monitoring methods, such as manual counting, pneumatic road tube, fixed sensors and loop detectors or cameras come with substantial costs and logistical constraints. These systems typically require significant installation and maintenance efforts. Moreover, their static nature limits their ability to adapt to dynamic traffic conditions as well as providing real-time data across large urban areas, compromising its effectiveness.

In this context, the emergence of drone technologies presents a revolutionary approach to traffic monitoring. They offer a versatile and cost-effective alternative for real-time traffic management. Equipped with advanced imaging technologies and smart algorithms, drones could dynamically monitor traffic flow, reacting quickly to perturbations that lead to congested areas by adapting the traffic mechanisms around the relevant area.

The advantages of utilizing drones for traffic monitoring lie in their mobility and flexibility. Unlike fixed sensors, drones can be deployed instantly to any location within the urban grid, capturing with an overall view the actual traffic conditions in any area, as well as its adjacent perimeter. They can capture high-resolution images that can be processed using advanced algorithms to extract critical traffic parameters and patterns. These data can be used to optimize traffic signal timings, improving overall traffic management strategies by developing predictive models and smart algorithms.

In essence, the primary objective of our project is to conduct a simulation of a drone fleet operating in a grid-like city, where zones of importance light up cyclically, and drones are tasked with moving to these areas to capture relevant informations. One thing to note is that different zones light up which have different levels of importance. Small perturbations that represent minor incidents within the urban grid might not be interesting to monitor often. To achieve this, we have developed smart, rule-based algorithms that direct drone behaviours based on specific conditions. Our approach involves several key components :

- Simulation of the urban environment: We have created a simulated grid-like city where zones of importance light up in a cyclic manner. This environment mimics real-world urban dynamics, allowing us to test and refine our drone algorithms under simplified conditions.

- Rule-based algorithms: The drones are guided by rule-based algorithms that issue commands based on pre-coded conditions. Those algorithms ensure that drones can effectively move to and monitor the zones of importance as they light up, capturing the most important informations.

- Efficiency evaluation: Two metrics are proposed to quantify the performance of the algorithms.

- Patrolling behavior: Beyond simply responding to the highest importance, our drones are programmed to adopt a patrolling behavior. This ensures they do not remain stationary in one zone, but rather continue to move and monitor different parts of the city, thereby maximizing coverage and information capture.

- Reinforcement learning model: Our ultimate objective is to develop a reinforcement learning model that enables drones to learn and adapt their patrolling patterns autonomously. By integrating reinforcment learning, we try to enhance the efficiency of the hardcoded algorithms. This model should allow drones to identify and adapt to patterns within the grid, improving their overall perfomances and responsiveness to dynamic changes in the environment.

# 1 Simulation set-up

## 1.1 Environment

The environment in which the drones can move is a grid of 20x30 squares. In order to simulate some kind of events happening, e.g., accidents, red lights or just congestion, a value of importance between 0 and 1 is assigned to each node. Formally, on the grid, the importance of each square $(x, y)$ at time $t$ is given by $V(x, y, t) \in [0, 1]$. Two types of perturbation that contribute to the importance of each field have been modeled. The big perturbations are meant to be long-lasting and intense importance-wise. They are supposed to model an important event happening in the city, which could be an accident for exemple. The small perturbations are smaller and periodic. They, on the other hand, are supposed to model some unimportant event like a traffic light creating congestion regularly. Both types of perturbation are governed by an amplitude implemented as follows:

- Amplitude of the big perturbation:

$$A_b(t) = e^{\frac{-t}{0.7 \cdot T}} \tag{1}$$

- Amplitude of the small perturbation:

$$A_s(t) = \sin\left(2 \cdot \frac{5\pi}{T} \cdot t\right) \tag{2}$$

where $T$ is the total number of steps chosen for the simulation. Each perturbation is modeled as a 2D Gaussian centered somewhere on the map. Hence, the importance value of a particular field $(x, y)$ at time t can be expressed as :

$$V(x, y, t) = \sum_i A_i(t) \cdot e^{-\frac{1}{2}[(x-x_i)^2+(y-y_i)^2]} \tag{3}$$

with $(x_i, y_i)$ the coordinate of the perturbation $i$. What can be deduced from this function is that the further away you are from the perturbation, the less impacted you will be.

## 1.2 Drone fleet

We use the term fleet to highlight the fact that the drones do not act as individuals but as a team. Each drone in the fleet that patrols the environment can do several things. They can assess the significance of the four nodes around it (value of importance) and move to one of those based on an algorithm (up, down, left, right, stay).

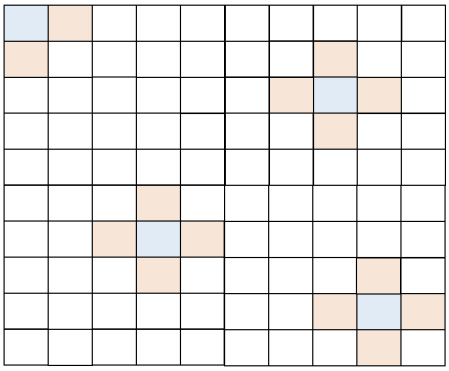

Figure 1: Adjacent nodes representation

Blue fields represent the actual positions of the drones while in orange one can see the adjacent nodes.

## 2 Classes

To develop this simulation, we chose Python as our programming language. Additionally, the extensive libraries available in Python provide many predefined functions and algorithms, greatly simplifying the coding process, such as NumPy, PyTorch, and Matplotlib.

To construct a comprehensive system for managing drones, we needed to establish a clear hierarchy among the Python classes.

The following figure illustrates the general framework of our project.

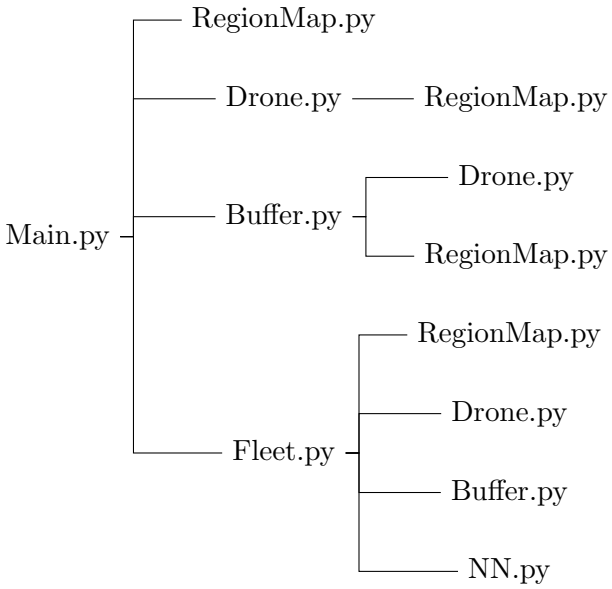


Figure 2: General framework of the python script classes hierarchy

The Main.py script serves as the central controller for the entire simulation. It is responsible for initializing key variables, such as the epicentre of the perturbations as well as the inital position of the drones and managing the tasks related to movement and interaction of the drones within the fleet. Below is a detailed breakdown of the scripts:

**Initialization**

- **Simulation parameters**
  - total number of times (iteration) in the simulation ($T$).
  - Map size ($x$ & $y$ sizes).
  - The position of Drones and perturbations.
  - Define areas of varying importance by calling the relevant function (RegionMap.py class).
  - Fleet creation.
  - Configuration of the structure for saving results which will be crucial for the RL part.

**Simulation Loop**

The main part of the Main.py script is composed of a loop that runs for the duration of the simulation $T$ which we defined earlier.

- **Data collection**
  - Collect observations & drone positions before and after movements.

- **State management**
  - Retrieves the state of the system which is composed of the positions of the drones as well as the adjacent nodes temporal values, and another metric not introduced yet that takes into account the time when a node was last visited. A state prime is also collected which corresponds to the state after the drones have taken an action.
- **Drone movement**
  - Take a decision based on the algorithm specified and calling the relevant function.
- **Reward calculation**
  - Based on the action taken, a reward is calculated. The reward given depends on several criteria that we will introduce later.
- **Buffer management**
  - It is a repository where data are processed and saved. The states and the rewards are saved by calling the relevant functions in the Buffer.py class.
- **Reinforcment learning**
  - RL will be deeply explained later, however when we chose to activate the decision-making by RL, batches are sampled from the buffer. Then, the neural network is pre-trained for a few epochs in an online learning manner, in an attempt to help improve drone decision-making.
- **Drones plotting**
  - Before each next iteration, a plotting of the fleet trajectories is executed, interactivally visualizing drone movements and their respective paths.

**Saving results**

At the end of the simulation, the arrays (which are a term for vectors in NumPy) responsible for data storage are saved for later pre-training of the reinforcment learning algorithm.

## 2.1 RegionMap

The RegionMap class is a critical component of the simulation and is responsible for representing and managing the environment in which drones operate. At each iteration, the 600 nodes that compose the 20x30 grid are dynamically updated. These nodes have values ranging between 0 and 1, where a value of 1 represents the maximum level of congestion, and a value of 0 indicates that there is no congestion, thus reflecting fluid traffic flow. This process effectively simplifies the simulation of real-world conditions in which traffic behavior and commuter patterns are constantly changing.

The variation in congestion levels across the grid is expressed using different mathematical models to simulate various types of perturbations:

**Small perturbations**

- They are modeled using a sinusoidal function to represent their cyclic nature, such as periodic congestion caused by red lights.
- The amplitude of small perturbations, $A_s(t)$ , varies quickly and is given by equation (2).

- This formula helps capture the rapid and periodic changes typical of minor traffic interruptions. The period of the small amplitude corresponds to $P = \frac{T}{5}$, which means that it completes five full cycles over the course of the simulation. Additionally, since $\sin(x) \in [-1, 1]$ , we need to ensure that the values of the importance remain non-negative. Therefore, we impose a condition such that if the value is less than 0, it is set to 0. Formally this can be expressed as :

$$A_s(t) = \begin{cases} 0 & \text{if } \sin\left(2 \cdot \frac{5\pi}{T} \cdot t\right) < 0 \\ \sin\left(2 \cdot \frac{5\pi}{T} \cdot t\right) & \text{if } \sin\left(2 \cdot \frac{5\pi}{T} \cdot t\right) \geq 0 \end{cases}$$

This adjustment ensures that $V(x, y, t)$ remains within the interval $[0, 1]$, effecitvely modeling the periodic nature of minor perturbations without allowing negative values.

### Big perturbations

- These perturbations are intended to simulate significant, long-lasting events such as accidents, which have a more prolonged impact on traffic flow.
- The amplitude of big perturbations $A_b(t)$ decreases slowly over time and is modeled by a negative exponential function given in equation (1).
- This formula reflects the gradual dissipitation of the impact of major incidents over time.

The overall importance value at each node $(x, y)$ at any given time $t$ is determined by combining the effects of all perturbations. The importance value of each node at any given time $t$, $V(x, y, t)$ is calculated using equation (3).

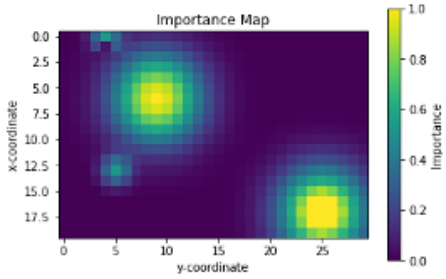


Figure 3: T=0

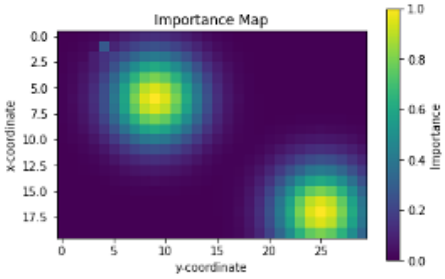


Figure 4: T=1

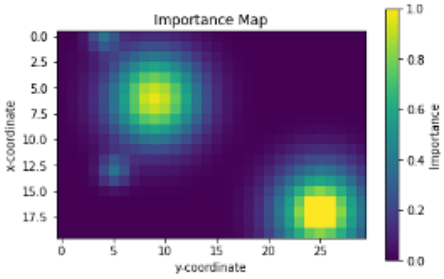


Figure 5: T=2

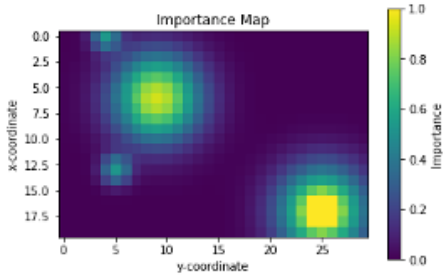


Figure 6: T=3

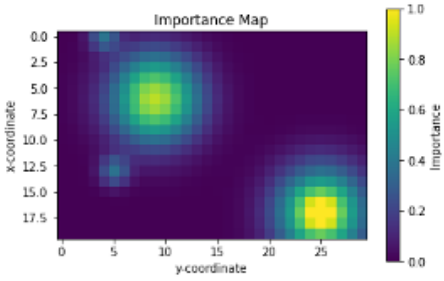


Figure 7: T=4

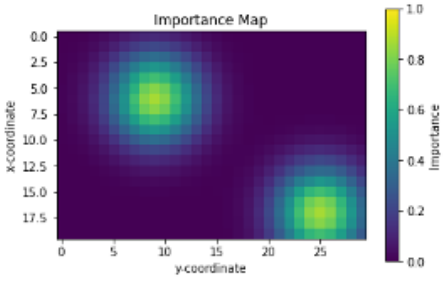


Figure 8: T=5

Figures 3 to 8 illustrate the different configurations of the map during the first five iterations of the simulation. In these figures, we can observe the cyclic behavior of the small perturbations, which are characterized by their periodic and rapid changes. Concurrently, the large perturbations exhibit a slow decrease in their intensity over time, reflecting their prolonged and gradual dissipation.

## 2.2 Drones

The drone class is responsible for various crucial things within the simulation. Indeed, each drone is controlled individually within the class, allowing them to collect information exclusively about their own state.This includes their current position as well as that of the nodes adjacent to them. With the drone in position $(x, y)$, the temporal importance $V(x, y, t) \in [0, 1]$ is also collected.

Moreover, this section of the Python script encompasses all the simple algorithms mentioned in the next section. Specifically, it implements the greedy algorithm, the random algorithm and the explorer algorithm. The command to move each drone is also executed within this class. Additionally, the feasibility of each action is assessed here ; whenever a drone is positioned near the edges of the map, certain actions become unfeasible and are consequently removed from the list of feasible actions.

## 2.3 Fleet

The Fleet class brings together individual drones to form a unified group that interacts with each other. This interaction involves sharing information such as position and the temporal importance of adjacent nodes. Consequently, drones can make informed decisions based not only on the immediate nodes around them but also on the information provided by other drones, covering all adjacent nodes within the fleet. In doing so, they are aware of perturbations occurring outside their field of vision.

We encountered an issue when attempting to transmit a certain intelligence to our drones. Specifically, drones lacked temporal memory, meaning that areas visited a short time ago would be re-visited immediately, yielding inefficiency. To address this, we introduced an additional map, referred to as the "mask map". This map serves as a multiplier for the initial importance map, effectively diminishing the importance of nodes that have been recently visited. This map allows us to give the drones some shared information through direct values. It is an easy solution to make our rule-based algorithms slightly more intelligent.

The mask map is initialized with the same dimensions as the initial importance map, specifically 20 x 30, where each field has a value equal to 1. When a node is visited, its value is reduced by a factor $a$. Additionally, the values of adjacent nodes are reduced by a factor $b$. Each node of the mask map regenerates gradually, with a value $c$ added to each node at each iteration. This mechanism ensures that drones can revisit a location if it remains important after some time has passed. Formally, this process can be described by the following expressions.

$$\text{Drone position} : V_{mask}(x, y) = V_{mask}(x, y) \cdot a \tag{4}$$

where $(x, y)$ are the coordinates of the mask map where the drone is currently located, and the value is multiplied by the factor $a$ to significantly decrease its value due to the visit.

$$\text{Adjacent nodes} : V_{mask}(x_{adjacent}, y_{adjacent}) = V_{mask}(x_{adjacent}, y_{adjacent}) \cdot b \tag{5}$$

$(x_{adjacent}, y_{adjacent})$ are the coordinates of the adjacent nodes in the mask map. Note that for each position of the drone, there are typically four adjacent nodes in Cartesian coordinates :

$$(\mathrm{X}_{\mathrm{adjacent}}, \mathrm{Y}_{\mathrm{adjacent}}) = [(x-1, y), (x+1, y), (x, y-1), (x, y+1)] \tag{6}$$

However, when the drone is positioned next to an edge, the number of adjacent nodes is reduced as some nodes fall outside the boundary.

$$\text{Map regeneration: } V_{\mathrm{mask}}(x, y) = \begin{cases} V_{\mathrm{mask}}(x, y) + c & \text{if } V_{\mathrm{mask}}(x, y) < 1 \\ 1 & \text{if } V_{\mathrm{mask}}(x, y) \geq 1 \end{cases} \tag{7}$$

This ensures that the temporal importance of each node is gradually restored, allowing drones to revisit locations that regain importance over time.

# 3 Simple algorithms

Simpler algorithms lay a benchmark for the analysis of the performance of the agent we wish to implement through reinforcement learning. Three of them have been implemented, i.e., random agent, greedy agent, and slightly less stupid (SLS) agent.

## 3.1 Metrics

Let us now introduce the metrics which will help us understand the quality of the two properties of each algorithm implemented: the patrolling metric and the importance metric.
A critical aspect of patrolling with a fleet of drones is the comprehensive coverage of the map, regardless of the specific importance of different areas. To quantify this, we introduce the patrolling metric. This metric is defined as the ratio of the number of nodes visited ($N_v$) by the fleet to the total number of nodes ($N$) on the map, which, in our scenario, is 600.

$$M_p = \frac{N_v}{N} \cdot 100[\%] \tag{8}$$

The other metric aims to measure the importance accumulated by the fleet during the simulation. While the importance value is completely deterministic, and we therefore have some not so interesting values at the end of the simulation, this metric does its job very nicely when we compare the different algorithms results, as we will see later on. It is formulated as follows:

$$M_{imp} = \sum_{t=1}^{T} \sum_{i}^{n} V(x_i, y_i, t) \tag{9}$$

with $n$ being the number of drones and $T$ the total number of timesteps of the simulation. This metric simply sums the importance value of the squares that the drones cover at each timestep.

## 3.2 Random Agent

This is the most simple algorithm. The action chosen by each of the drones is governed by randomness. This agent gives a very mediocre result, but provides an excellent baseline for the other algorithms. Though, in our simulation, a drone cannot stay more than 3 times in the same square.

## 3.3 Greedy Agent

The Greedy agent we have implemented follows the simple rule of moving (or staying!) to the node where the importance is maximal. The greedy agent also has only one constraint which is that he cannot stay more than three times on the same node.

## 3.4 Explorer agent

The explorer agent is a simple algorithm that follows a predefined hardcoded algorithm. Its path aims to patrol the entire map, visiting every node of the map.

## 3.5 SLS Agent

We developed a slightly less stupid agent that controls the four drones. With this agent, the goal was to increase patrolling while still getting some great results for the importance metric. This algorithm has the ability to give each drone an algorithm. For example, two drones could be driven by the greedy algorithm while the two others could be controlled by the random agent and the explorer agent.

### 3.6 Mask map/Shared information

In order to make the rule-based algorithms slightly more inteligent, we introduce shared information each drone has available. The goal is to be able to tune parameters to change the behavior of the drones while keeping the same rule-based algorithm. To illustrate the impact of the mask map on our algorithms, we have conducted a graphical comparison of the two metrics: one without the mask map and one with the mask map. Below are the graphs for each of the metrics. The parameters of the mask map for this greedy agent simulation are as follows:

$$a = 0.005, b = 0.3, c = 0.0001$$

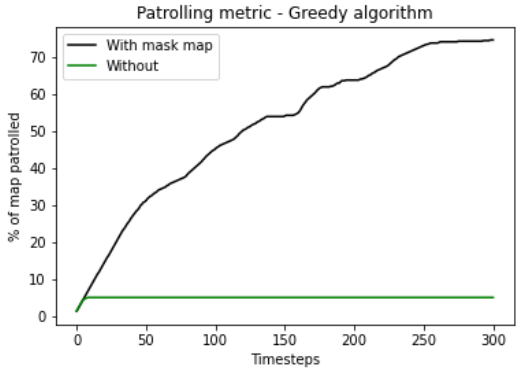


Figure 9: Effects of the mask map - Patrolling metric

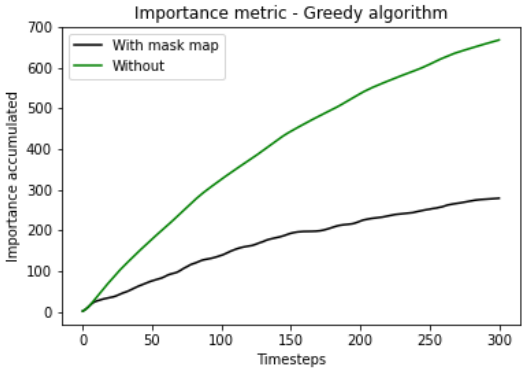


Figure 10: Effects of the mask map - Importance metric

In figure 9, the patrolling metrics demonstrate a significant improvement when the mask map is introduced. The temporal memory mechanism is effectively functioning, as the drones do not remain in the same zone for the duration of the simulation, which was the case prior to introducing the mask map. Consequently, patrolling efficiency has increased, though this comes at the cost of a considerable reduction in the accumulated importance. This outcome was to be anticipated ; as the drones now have a "modified" importance when the mask map is superimposed on the importance map, resulting in the highest zones of perceived importance not being the highest in reality. This occurs because the mask map artificially reduces the importance of the zones to force the drones to patrol the entire map.

The two graphs display separate curves for each metric: the patrolling metric and the importance metric. The mask map, as described above, significantly enhances the patrolling efficiency of the greedy algorithm. This improvement is credited to the "modified" importance values when the mask map is superimposed on the importance map, which substantially alters the drones' behavior, promoting a patrolling pattern. The drones thereby acquire the temporal memory they previously lacked. The following figures illustrate these differences, showing the respective paths of the drones for the greedy algorithm under the two different configurations.

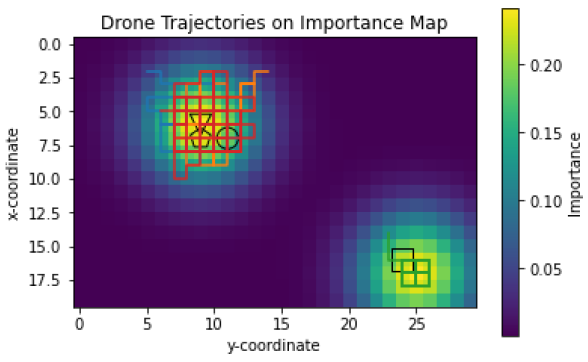


Figure 11: Greedy algorithm without mask map - path of the drones

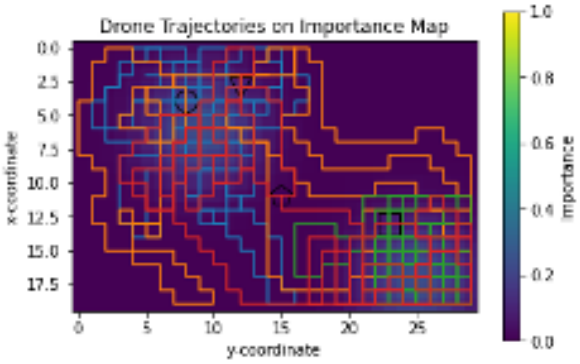


Figure 12: Greedy algorithm with mask map - path of the drones

Without application of the mask map, modifying the perception of importance, the drones tend to converge in the same areas, leading to inefficient behavior. Specifically, in the bottom right corner of the map, a single drone repeatedly patrols a small area along the identical path throughout the simulation. Meanwhile, the remaining three drones focus on a significant perturbation area located in the top left corner of the map for the same duration. Although their paths are not exactly identical, the overall patrolling strategy proves itself ineffective, as the drones fail to cover any zones of importance beyond the major perturbation.

Contrarily, as illustrated on Figure 12, the incorporation of the mask map significantly enhances the drones 'patrolling efficiency'. The drones exhibit an improved and effective patrolling behavior by traveling to zones outside the range of the usual perturbations. This improvement aligns with the objective of introducing the mask map. From a traffic management perspective, this behavior implies that the drones are capable of patrolling the city and are able to respond quickly to unusual incidents. With this enhanced patrolling strategy, drones are strategically positioned so that not all units are concentrated in one area. Consequently, they can fastly react and move to significant location outside the usual perturbation zones.

# 4 Simulation using the simple algorithms

We have conducted simulations using different agents. The mask map parameters for the greedy agent are as follows :

$$a = 0.3, b = 0.5, c = 0.05$$

These values differ from the parameters previously utilized, display in Figure 12. The relatively high values for $a$ and $c$ promote a greedy behavior in the drones, which is the case in this simulation.

Concerning the SLS agent, we have configured two drones to exhibit a greedy behavior, while one drones explores the map by following a predefined path and another drone follows a random path.

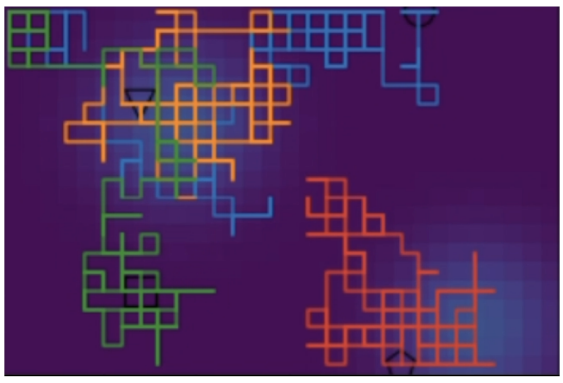

Figure 13: Random agent

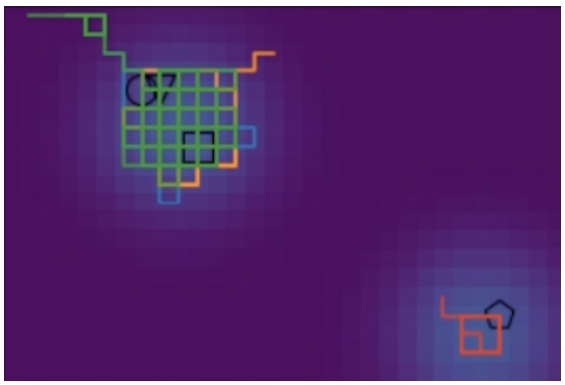

Figure 14: Greedy agent

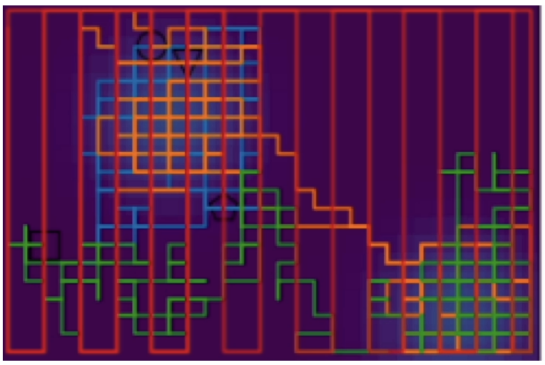

Figure 15: SLS agent

The Figures 13 to 15 represent the paths of drones using the three different algorithms.

Figure 13 illustrates the paths taken by drones using the random algorithm :

- The paths appear scattered and distributed throughout the map.
- This method provides a great coverage of the map, reducing the likelihood of neglecting any particular area.
- However, due to their randomness, drones may not prioritize areas of greater importance, which can lead to slower response times for incidents in critical zones.

In Figure 14, the greedy algorithm is utilized resulting in the following observations :

- Due to the parameters of the mask map, the drones tend to converge on areas with the highest importance values. While there is significant overlaping in the paths, the covering of the less important areas is strong.
- While this approach ensures that high-importance zones are frequently patrolled, it results in neglecting other areas.

In Figure 15 the SLS algorithm is implemented with two drones following greedy paths, one following a random path, and one exploring along a predefined path:

- This approach shows a balanced distribution of paths, combining the benefits of both greedy and random algorithms.
- The greedy drones ensure that high-importance areas are regularly patrolled, while the random and predefined path drones contribute to wider coverage of the map.
- This strategy enhances the overall efficiency by maintaining vigilance in critical zones while also covering less important areas, thereby improving response times to incidents occuring outside the main zones of importance.

### 4.1 Results of the simulation

The metrics of these simulations are displayed in Figure 16 and 17. However, we modified the parameters of the mask map with the following :

$$a = 0.005, b = 0.3, c = 0.0001$$

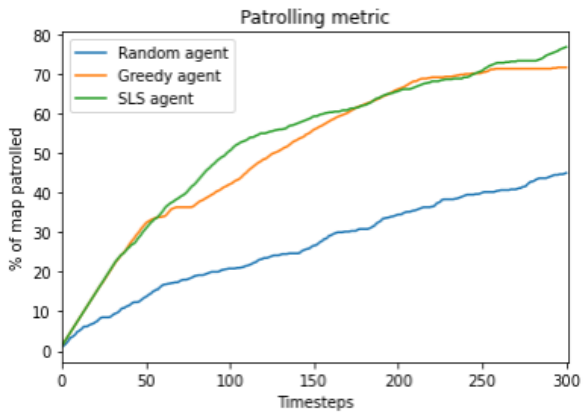


Figure 16: Patrolling metric of the algorithms

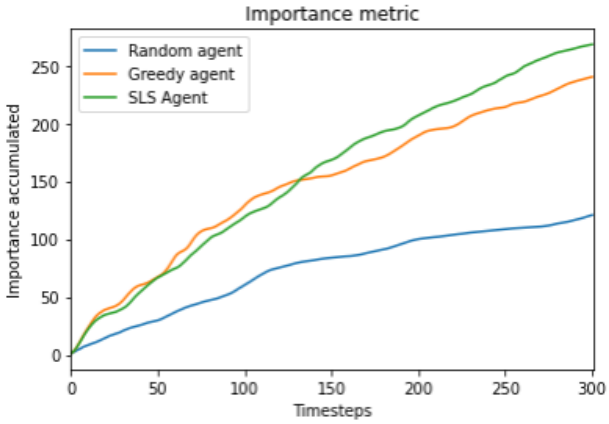


Figure 17: Importance metric of the algorithms

In Figures 15 and 16, it is evident that the performance of the random agent is unsatisfactory. The random agent performs poorly on both metrics, making it the least effective algorithm in terms of performance. However, it remains valuable for collecting diverse data for the reinforcment learning (RL) agent. The SLS agent and the greedy agent with the mask map exhibit similar performance levels, although the SLS agent performs slightly better. This outcome aligns with our expectations, as the SLS agent was designed to achieve superior results compared to the greedy agent. Notably, the greedy agent achieves strong patrolling metrics, highlighting the importance of incorporating the mask map for effective performance.

The comparison of these three algorithms reveals distinct strenghts and weaknesses. The greedy agent focuses on critical areas but may lack comprehensive coverage on some situations. The SLS algorithm strikes a balance, leveraging the strengths of both approaches to achieve effective patrolling and coverage. This hybrid approach ensures that drones can quickly respond to incidents across the entire map, making it the most efficient and effective strategy among the three.

# 5 Reinforcement learning

## 5.1 General framework

Reinforcment learning (RL) is a type of machine learning approaches in which an agent learns to make decisions by interacting with an environment. The fundamental goal of RL is for the agent to develop a strategy, or policy, that maximizes the cumulative reward over time. Here is an overview of the basic RL framework, including its key components and the process of learning :

**Key Components**

- **Agent**: The decision maker that interacts with the environment.
- **Environment**: The external system with which the agent interacts.
- **State (s)**: A wise selection of the data representing the current situation in the environment.
- **Action (a)**: The set of all possible moves or decisions the agent can make.
- **Reward (r(s,a))**: Feedback from the environment in response to the agent's actions, guiding learning.
- **Policy ($\pi$)**: A strategy that maps states to actions, directing the agents behavior.
- **Q-value (Q(s,a))**: A function representing the expected cumulative reward of taking a particular action in a given state.

The primary objective of RL is to determine an optimal policy $\pi$ that maximizes the expected cumulative reward over time. This involves finding the best strategy for the agent to follow, ensuring that it takes actions that lead to the highest possible rewards in the long run. The agent continuously interacts with the environment, receives feedback in the form of rewards, and updates its policy to improve decision-making capabilities over time.

In our work we employ a reinforcment learning technique known as Deep Q-Learning. Deep Q-Learning enhances traditional Q-Learning by utilizing a deep neural network to approximate the Q-value function, which predicts the expected rewards for actions in given states.

## 5.2 ReplayBuffer

The replay buffer is a crucial component in our reinforcement learning framework. This class is designed to store relevant data necessary for training our RL algorithm. Specifically, the replay buffer saves arrays of states, actions, rewards, and next states (denoted as $s'$) generated during simulations. By maintaining a record of these experiences, the replay buffer allows for more efficient and stable training of the RL agent, enabling it to learn from a diverse set of past experiences.

### 5.3 State

The state representation for each drone at each timestep is composed of the following elements :

- **Drone position** The current position of the drone on the map, scaled to match the order of magnitude of the other state elements.
- **Scaled importance** $V(x, y, t)$ **:** The importance values of all possible next fields. These fields represent the potential next position the drone can take, specifically the importance of adjacent nodes.
- **Average Mask Value (**$V_{Mask}(x, y, t)$**:** The average mask value in all possible move directions. This is calculated as the average mask value of every field from the drone's current position to the edge of the map in each direction.

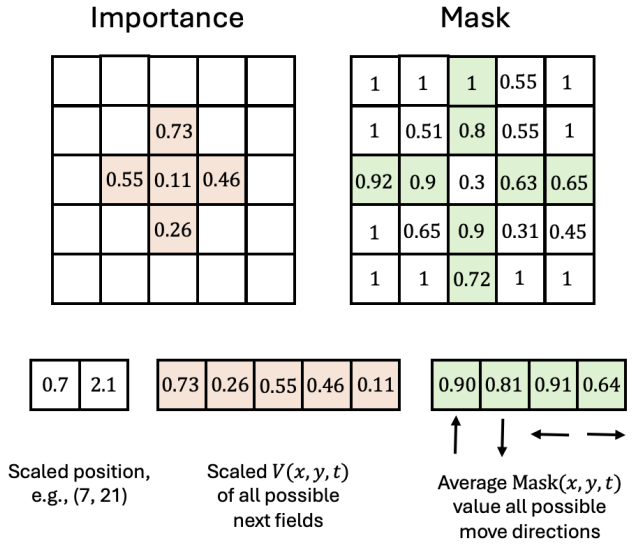


Figure 18: State representation

### 5.4 Reward

A reward is implemented and has two main goals : rewarding and encouraging good actions while penalizing and discouraging bad actions. Its expression is as follows :

$$R(s, a, s') = \alpha_1 \left( \sum_{\substack{x',y' \\ \in \text{ adj}}} V(x', y', t+1) - \sum_{\substack{x,y \\ \in \text{ adj}}} V(x, y, t) \right) + \alpha_2 \left( \sum_{\substack{x',y' \\ \in \text{ adj}}} V_{mask}(x', y', t+1) - \sum_{\substack{x,y \\ \in \text{ adj}}} V_{mask}(x, y, t) \right) \tag{10}$$

where :

- $R(s, a, s')$ is the reward for taking action a in state $s$ and transitioning to state $s'$.
- $\alpha_1$ and $\alpha_2$ are weight parameters that balance the contribution of the two components of the reward function.
- $V(x, y, t)$ represents the importance value at position $(x, y)$ at time $t$.

- $V_{mask}(x, y, t)$ represents the mask value at position $(x, y)$ at time $t$.
- $V(x', y', t+1)$ represents the importance value at the next time step $t+1$ at position $(x', y')$
- $V_{mask}(x', y', t+1)$ represents the mask value at the next time step at position $(x', y')$

**Detailed explanation** The reward function is composed of two main components :

- Greedy reward component: This is represented by the first term of the equation, which is multiplied by $\alpha_1$. This component measures the change in importance before and after taking an action. Specifically, it calculates the sum of importance values at adjacent positions $(x', y')$ at the next time step $t+1$ and substracts the sum of importance values at adjacent positions $(x, y)$ at the current time step $t$. The difference between these sums indicates whether the agent (e.g., a drone) has moved to a more important area. A positive difference signifies that the agent has relocated to a region with higher overall importance.
- Patrolling reward component : This is represented by the second term of the equation, which is multiplied by $\alpha_2$. This component measures the change in mask values before and after an action is taken. It computes the sum of mask values at adjacent positions $(x', y')$ at the next time step $t+1$ and substracts the sum of mask values at adjacent positions $(x, y)$ at the current time step $t$. The difference between the two sums indicates whether the agent has moved to an area that has not been visited recently. A positive difference means the agent has moved to a position that was "less explored" than the previous position.

The reward function significantly influences the behavior and learning process of the reinforcment learning agent. Adjusting the weights $\alpha_1$ and $\alpha_2$, the balance between the greediness of the agent and the patrolling behavior can be controlled. This reward function is designed to guide the agent towards achieving an optimal balance between moving to important areas and patrolling the map. These parameters provide flexibility in how the importance and patrolling behaviors are weighted and thus influence the agent's learning process accordingly.

### 5.5 Action

In deep Q-learning, a deep neural network (DQN) is used to approximate the Q-value function. The network takes the current state as input and outputs Q-values for all possible actions. The action with the highest Q-value is chosen during the RL simulation. Below is displayed the architecture of our Q-network

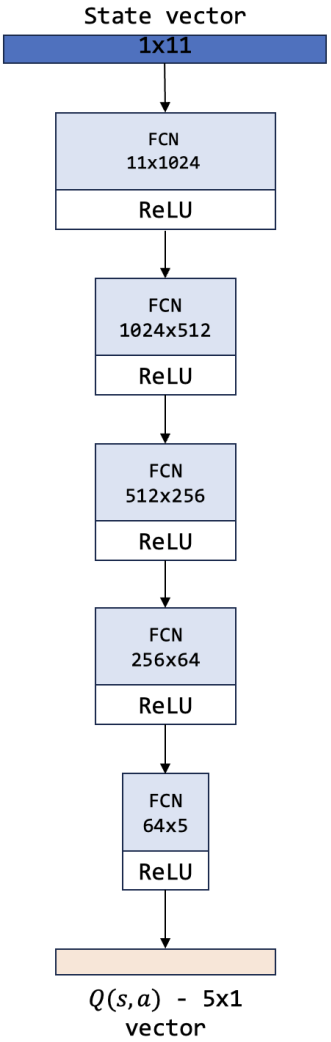


Figure 19: Network structure

### 5.5.1 DQN Learning

In our algorithm, we use two neural networks: the policy network and the target network. The policy network estimates the Q-values and is updated frequently. The target network, a copy of the policy network, is updated less frequently and provides stable target Q-values for training the policy network. In DQN, backpropagation is performed by first randomly sampling a batch of experiences $(s, a, r, s')$ from the replay buffer. For each experience, the target Q-value $y$ is calculated using the Bellman equation:

$$y = r + \gamma \max_{a'} Q_{\text{target}}(s', a') \tag{11}$$

The policy network predicts $Q_{\text{policy}}(s, a)$ for the current state and action. The loss is computed as the mean squared error between $y$ and $Q_{\text{policy}}(s, a)$. Backpropagation is then used to update the policy network's weights by minimizing this loss through gradient descent. This approach with two networks stabilizes the learning process by reducing the correlations between consecutive updates and providing more reliable training targets.

$$\text{Loss} = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q_{\text{policy}}(s_i, a_i))^2$$

where $N$ is the size of the mini-batch.

### 5.6 Pre-training

To expedite the convergence of the reinforcement learning (RL) process, we employed our rule-based algorithms for pre-training. Specifically, we gathered data from the random, greedy, and SLS (Slightly Less Stupid) agents, which was subsequently utilized in the pre-training phase. This pre-collected data served as input for the RL agent, enabling it to learn patterns and pre-tune its parameters before making decisions independently.

By leveraging this pre-training strategy, the RL process achieved faster convergence. The pre-training phase allowed the RL agent to develop an initial understanding of the environment and decision-making patterns, thereby reducing the time required for the agent to optimize its policy during actual training.

Below, we present the pre-training loss results for two different learning rates. These results demonstrate the effectiveness of the pre-training approach in accelerating the learning process and improving the overall efficiency of the RL algorithm.

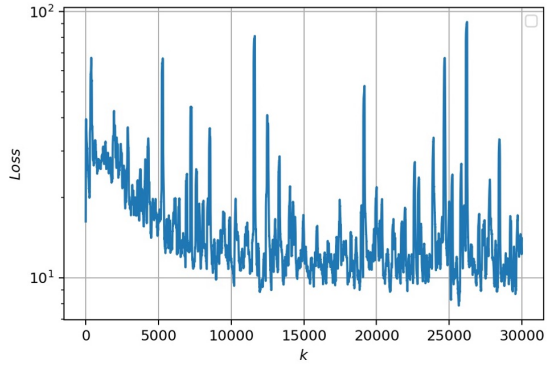


Figure 20: Phase 1 - pre-training with $l_r = 10^{-3}$

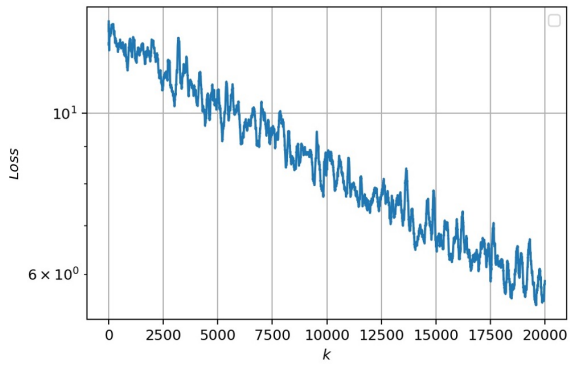


Figure 21: Phase 2 - pre-training with $l_r = 10^{-4}$

- **Phase 1 , $l_r = 10^{-3}$ :** The loss starts relatively high, indicating the initial error in the model prediction. Overtime, the loss generally decreases, suggesting that the model is learning and improving its prediction. However, the large and frequent spikes indicate the instability of the learning process. This could be due to the high learning rate causing the model to take large steps, leading to overshooting and high variance in the loss. The loss does not settle smoothly, and high fluctuations persist even toward the end of the training period of $k = 30'000$.

- **Phase 2, $l_r = 10^{-4}$ :** Similar to the first phase, the loss starts at a high value, reflecting initial errors. Fluctuations in the loss values are less pronounced compared to the higher learning rate. Moreover, the loss decreases smoothly over time, indicating a more stable learning process. The smaller learning rate allows the model to take smaller, more controlled steps, reducing the likelihood of overshooting and resulting in more gradual improvement. Finally, the loss continues to decrease steadily, suggesting that the model achieves a more stable convergence.

# 6 Results and Discussion

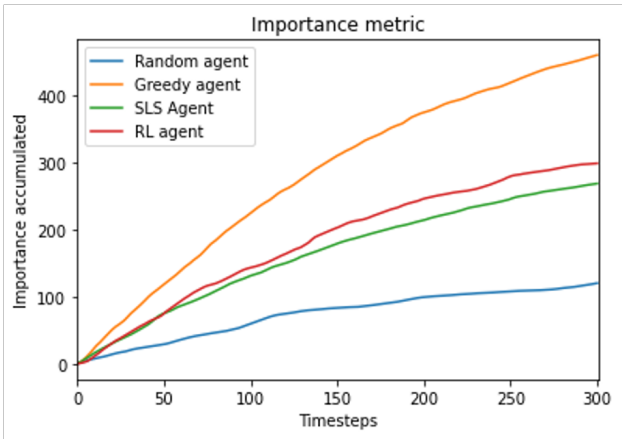


Figure 22: Importance metric

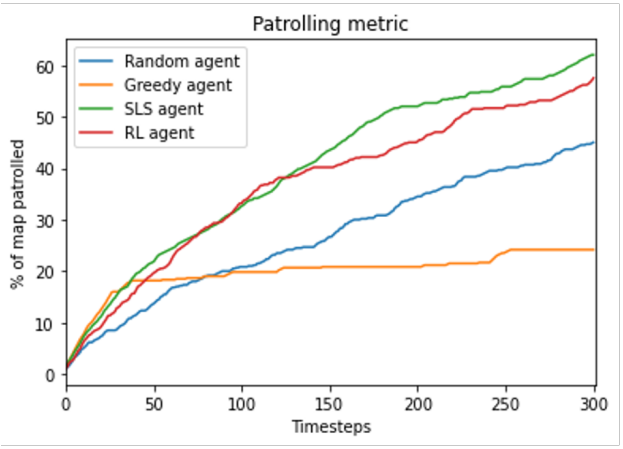


Figure 23: Patrolling metric

It is possible to observe that the RL agent is 2nd in each metric. This is a promising result as the greedy algorithm is designed to be greedy and therefore have as much possible importance, and the SLS agent is designed with a solely patrolling drone, and those algorithms are first in each metric respectively. This is quite logical given our simulation setup.

For the reinforcement learning, playing around with the state fed to the networks may yield even nicer results. We have seen during our project that there are certain structures that struggle to capture what needs to be learned. Of course, the hyperparameters like the learning rate play a big part in the convergence, but sometimes the networks just won't converge according to the Bellman equation, e.g. we tried to capture the features of the map with a convolution network, but the pretraining was inconclusive.

Taking this into account, the fact that we were able to obtain some kind of convergence during our pre-training is highly encouraging. As we developed the whole simulation from scratch (map, importance, reward, architecture of the network), this wasn't a certainty. But those results are promising, and it could be very interesting to apply our work to real world data. Of course, with more time, there are a lot of things whose influence could be more thoroughly investigated, such as the hyperparameters, the mask map, the buffer, the state etc.

# 7 Conclusion

In this project, we aimed to explore the potential of using a fleet of drones for effective traffic monitoring in urban areas through the implementation and evaluation of various algorithms, including reinforcement learning techniques. Our simulation model successfully demonstrated how the different algorithms we coded from scratch could guide drone behaviors to optimize traffic monitoring.

Our simulation began with the creation of a simplified urban environment described by a grid. Each field of the grid was assigned a dynamic importance value to replicate real-world traffic perturbations. This setup allowed us to test and refine our drone algorithms under controlled conditions.

We developed and tested several rule-based algorithms that all exhibit their strengths and weaknesses ; the random agent provides wide coverage but lacked focus on high-importance zones, the greedy agent effectively prioritizes critical areas at the expense of a great coverage while the SLS agent balanced the strength of both approaches, by adding an explorer path. This last algorithm ensures effective patrolling and coverge coupled with a great greedy behavior that focuses on critical areas by combining different algorithms for the different drones.

To enhance the perofrmance of our rule-based algorithms, we introduced the concept of a mask map. This map acted as a temporal memory, preventing drones from revisiting recently monitored areas too frequently. The mask map significantly improved patrolling efficiency and ensured a more uniform distribution of drone activity across the map.

Furthermore, we incorporated reinforcement learning through Deep Q-Learning to enable the drones to learn and adapt their patrolling patterns autonomously. The reinforcement learning agent showed promising results. It demonstrates the potential of reinforcement learning to improve the efficiency of traffic monitoring systems. In addition, a hybrid approach, using the strengths of both rule-based algorithms and reinforcement learning, can provide an effective strategy using a fleet of drones to monitor traffic.

Despite the satisfactory success of our simulations, there are several paths of improvement. Fine-tuning the hyperparameters, exploring different neural network architectures, experimenting with different feeding data to the network, exploring diverse kinds of neural network architectures, and integrating real-world data could further improve the system's performance. Additionally, extending the simulation to larger and more complex urban environments would provide a deeper evaluation of the methods proposed.

To conclude, our project demonstrates the feasibility and potential use of a coordinated fleet of drones for dynamic and efficient traffic monitoring. The combination of various algorithms and reinforcement learning offers a robust framework for real-time traffic management, capable of adapting to the ever-changing conditions of urban transportation networks.

# 8 Acknowledgments

We would like to express our gratitude to our supervisor, Marko Maljkovic, for his tireless support and guidance throughout the semester. His availability and invaluable assistance during moments of struggle were crucial in the successful completion of our Bachelor Thesis. Marko's expertise and encouragements greatly contributed to our understanding and development of the simulation. We are deeply thankful for his dedication and mentorship over the course of the project.

Thank you Marko.

Marius, Léo and Oscar.