Università degli Studi di Milano
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

CORSO DI LAUREA TRIENNALE IN
INFORMATICA

MULTI-ROBOT PATROLLING WITH HETEROGENEOUS UAV/ASV FLEETS FOR WATER MONITORING GUIDED BY A GENERATIVE VAE-UNET

Supervisor: Prof. Nicola Basilico
Co-supervisor: Dott. Michele Antonazzi

Elaborato Finale by:
Edoardo Rodiani
Matr. No. 964881

Academic Year 2023-2024

a voi che all'inizio di questo percorso ve ne siete andati
ma più di chiunque altro avreste voluto esserci
ai miei nonni, Rosi, Adriano e Lillo

Ringraziamenti

Vorrei ringraziare il mio relatore Prof. Nicola Basilico, correlatore Dott. Michele Antonazzi e Prof. Matteo Luperto per avermi dato la possibilità di prendere parte allo sviluppo di questo progetto.

Ringrazio i miei genitori, Daniela e Fabio, per avermi supportato e sopportato fino al raggiungimento di questo traguardo. Durante la mia infanzia e adolescenza, mi hanno potuto offrire tutto ciò di cui avevo bisogno, nonostante i sacrifici che ciò potesse comportare. Permettendomi di crescere con la sicurezza e la forza necessarie per raggiungere questo traguardo. La loro presenza, i loro insegnamenti e la capacità di avermi lasciato gli spazi di cui avevo bisogno sono stati le fondamenta su cui ho potuto costruire questo risultato.

Ringrazio i miei parenti e i miei amici per il supporto morale e l'incoraggiamento che mi hanno dato. Per le giornate passate in compagnia che mi hanno permesso di ritagliare momenti di spensieratezza e svago fondamentali per affrontare questo percorso con più serenità.

Desidero esprimere un sentito ringraziamento speciale alla mia ragazza, Azzurra. Fin da prima di questo percorso è stata al mio fianco in ogni momento, sia nei giorni più luminosi che in quelli più difficili. È diventata il mio rifugio sicuro, un abbraccio in cui perdersi, un orecchio sempre pronto ad ascoltarmi e una voce con cui condividere sorrisi, lacrime e pensieri. Le sue parole, sempre ben scelte, sono state un faro che mi ha guidato, aiutandomi ad affrontare ogni sfida con la consapevolezza delle mie capacità, senza mai sottovalutarmi. Grazie per essere stata sempre al mio fianco come compagna di vita.

Infine terrei a ringraziare me stesso, per la forza e la determinazione dimostrate lungo questo percorso. A te che hai saputo dosare impegno e pazienza, perseveranza e speranza, devo questo traguardo.

Sommario

Questo documento estende un progetto precedente che affronta il Non-Homogeneous Patrolling Problem. La soluzione utilizza Autonomous Surface Vehicles (ASVs) con sensori per monitorare i Water Quality Parameters (WQP) in tempo reale tramite una rete Deep Reinforcement Learning (DRL) per la selezione delle azioni ASV, e una rete Variational Auto-Encoder UNet (VAE-UNet) per la stima della distribuzione totale dei WQP utilizzata come input per la DRL per migliorare l'apprendimento.

La novità spiegata in questo documento è l'introduzione di una flotta eterogenea composta dagli ASV precedentemente presenti e dai nuovi Unmanned Aerial Vehicles (UAVs) (droni) che scattano istantanee dall'alto, che cooperano per raggiungere lo stesso obiettivo del progetto precedente. Gli ASV pattugliano aree ad alto rischio, mentre i droni esplorano rapidamente zone non visitate (da sempre o da lungo tempo).

L'elaborato si concentra sull'implementazione dei nuovi componenti software che simulano l'ambiente e i movimenti dei droni. Si discute poi dell'implementazione di diversi filtri per simulare il rumore nelle misurazioni dei droni: un filtro senza rumore, uno che simula le limitazioni della fotocamera (effetto fish-eye) e uno per condizioni meteorologiche avverse (nuvolosità del cielo). Si passa poi a discutere l'addestramento e il test della VAE-UNet con i nuovi dati della flotta eterogenea e i vari confronti in vari scenari tra il vecchio modello omogeneo e quelli nuovi. I nuovi modelli eterogenei VAE-UNet superano le prestazioni del modello omogeneo con alcuni di questi filtri di rumore.

Considerando l'errore medio pesato con l'importanza delle aree superficiali come metrica. Rispetto al vecchio modello omogeneo, nello scenario migliore, senza rumore, c'è un miglioramento del 64%. Con rumore della telecamera, il miglioramento è del 34%. I risultati con rumore meteorologico sono subottimali ma, introdotti solo per testare le peggiori condizioni teoriche, la nuvolosità è improbabile a causa della bassa altezza di volo dei droni.

I confronti vengono effettuati tra i nuovi modelli e il precedente in scenari dinamici. Sebbene sia i nuovi modelli che quello vecchio sono stati addestrati solo su scenari statici, i nuovi modelli mostrano miglioramenti anche in contesti dinamici indicando una maggiore robustezza e fornendo un'apertura promettente per ulteriori sviluppi nella ricerca sul problema del pattugliamento con l'uso di flotte eterogenee con un obiettivo comune.

Summary

This paper extends a previous project that addresses the Non-Homogeneous Patrolling Problem. The solution uses Autonomous Surface Vehicles (ASVs) with sensors to monitor Water Quality Parameters (WQP) in real time via a Deep Reinforcement Learning (DRL) network for ASV action selection, and a Variational Auto-Encoder UNet (VAE-UNet) network for estimating the total WQP distribution used as input to the DRL to improve learning.

The novelty explained in this paper is the introduction of a heterogeneous fleet composed of the previously present ASVs and the new Unmanned Aerial Vehicles (UAVs) (drones) that take snapshots from above, that cooperate to achieve the same goal of the previous project. ASVs patrol high-risk areas, while drones quickly explore unvisited (forever or for a long time) zones.

The paper focuses on the implementation of the new software components that simulate the environment and the drone movements. The implementation of different filters to simulate noise in the drone measurements is then explored: a filter without noise, one simulating camera limitations (fish-eye effect) and one for adverse weather conditions (sky cloudiness). The training and testing of the VAE-UNet with the new data from the heterogeneous fleet and the various comparisons in various scenarios between the old homogeneous model and the new ones are then discussed. The new VAE-UNet heterogeneous models outperform the homogeneous model with some of these noise filters.

Considering the weighted mean error with the importance of surface areas as a metric. Compared to the old homogeneous model, in the best-case scenario, without noise, there is an improvement of 64%. With camera noise, the improvement is 34%. The results with weather noise are suboptimal but, introduced only to test the worst-case theoretical conditions, cloudiness is unlikely due to the low flight height of the drones.

Comparisons are made between the new models and the previous one in dynamic scenarios. Although new models and the old one were only trained on static scenarios, the new models also show improvements in dynamic contexts indicating greater robustness and providing a promising opening for further developments in research on the problem of patrolling with the use of heterogeneous fleets with a common goal.

Contents

Ringraziamenti i

Sommario ii

Summary iii

Contents iv

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Problem statement . . . 3
  - 1.2.1 DRL for ASV actions . . . 3
  - 1.2.2 Introduction to VAE-UNet in the project . . . 5
- 1.3 Heterogeneous fleets with common objective . . . 8

2 State of the art 9
- 2.1 Before VAE-UNet . . . 9
  - 2.1.1 Gaussian Process . . . 9
  - 2.1.2 K - Nearest Neighbors . . . 10
- 2.2 Before DRL . . . 10
  - 2.2.1 Particle Swarm Optimization . . . 10

3 Implementation of the new components 12
- 3.1 Introduction to heterogeneous fleets . . . 12
  - 3.1.1 Water quality parameters . . . 12
  - 3.1.2 Roles of the fleets . . . 13
  - 3.1.3 Cooperation between the fleets . . . 13
- 3.2 New components . . . 15
  - 3.2.1 Drone . . . 15
  - 3.2.2 CoordinatedHetFleet . . . 16
  - 3.2.3 NoiseModels . . . 19

3.2.4 HetModels . . . 21
3.2.5 DiscreteModelBasedHetPatrolling . . . 22
3.2.6 HetPathPlanners . . . 24
3.2.7 TimedDiscreteModelBasedHetPatrolling . . . 24

4 Experimental Evaluation **28**
4.1 Protocol . . . 28
4.1.1 Parameter setting . . . 28
4.1.2 Dataset creation . . . 30
4.2 Training . . . 31
4.3 Validation . . . 33
4.4 Testing . . . 35
4.5 Models put to test . . . 36
4.5.1 Static test . . . 36
4.5.2 Peaks variation test . . . 38
4.5.3 Dynamic test . . . 40
4.5.4 Dynamic Peaks Variation test . . . 42

5 Conclusions **45**
5.1 Conclusions . . . 45
5.2 Future developments . . . 45

Bibliography **47**

Chapter 1

Introduction

This introductory chapter briefly explains the already existing project, which forms the foundation of this paper, to provide the reader with important key information.

This chapter ends with the presentation of the new task that this paper will focus on.

The next chapters focus on the new task, its development and analysis.

1.1 Introduction

Lakes and shores play a vital role in the ecosystems of the earth and are critical components of our planet's water cycle.

The quality of water in these environments constantly changes based on facts such as rainfall patterns, temperature fluctuations and most importantly human activities such as agriculture and urbanization [1].

Monitoring water pollutants and disease-causing organisms is essential to understand and manage the impact on health.

One of the most common example of algae is the cyanobacteria (blue-green algae). As article [2] states, the problem is that during cyanobacteria blooms a large number of dead cyanobacteria sinks to the bottom and decompose, consuming oxygen. This leads to the reduction of dissolved oxygen in the lake affecting the living conditions of the ecosystem's fauna.

This blooms directly affect also drinking water. There are lots of examples described in article [2] of directly caused problems to many people's drinking water safety.

They also cause problems to flora and microorganisms of water bodies.

Traditional manual monitoring methods are not quite useful if the request is to collect comprehensive data on the changing conditions of water resources and cover large areas [3]. Autonomous Surface Vehicles (ASVs) are useful for this task [4]. With a good policy to coordinate them, they can continuously gather data on various physical and chemical

properties of water, providing a more comprehensive understanding of the risks these environments suffer.

The old project aims to address the problem of finding a good policy to move the ASVs that take measurements of water quality parameters (WQP) pursuing this objectives:

- obstacle avoidance
- efficient coordination between vehicles
- integration of data gathered by multiple vehicles into a comprehensive picture of water conditions

In the case of water quality monitoring the measurements have to be continuous. Already visited zones have to be revisited after some time because of the possible changes in the algae bloom distribution or because of the higher danger in a particular area. This is the main problem of the project: the Patrolling Problem [5] [6].

The distribution of blooms are different from one zone to another. This means that some zones have to be covered more rapidly than others. Essentially ASVs have to take in consideration the danger of a particular zone while still giving space to zones that were never visited even once. The problem here becomes a Non-Homogeneous Patrolling Problem (NHPP).

The project proposed to use Deep Reinforcement Learning (DRL) as other previous approaches that used it [6] [7] [8] shown promising results in generating tailored policies optimized for patrolling allowing ASVs to make decisions on the fly based on real-time interactions with the environment. However, there is a clear uncovered aspect of these methods when simultaneous modeling and patrolling are required, especially when involving multiple vehicles. More particularly the project uses Double Deep Q-Learning (DDQL) that essentially learns to estimate the reward of taking an action given also the state where the agent that is taking the action is at.

However, giving this model only the positions of the agents and what they actually read as input, isn't an accurate enough observation to infer optimal actions for agents, without a good observation input it is harder to infer actions for agents to move [6]. So to give this model a proper state to start from, another model was introduced: a Deep Variational Auto-encoder (VAE) based on popular UNet architecture [9] (VAE-UNet), to infer the complete distribution of algae based on only the measurements taken by the ASVs (partial observation of the total distribution) like in an Inverse Problem Estimation[10]. A much more comprehensive input information that highly enhances the DRL model understanding. The input to the VAE-UNet model is always the same size and takes the same time of inference no matter the moment when it gets used. A way better model than the most conventional: Gaussian Process or K-Nearest Neighbors.

The project from where the new task of this paper starts from introduces this solutions:

- A Deep Reinforcement Learning framework to obtain deep patrolling policies by means of an appropriate NHPP formulation, reward and state.
- A Variational Auto-Encoder model to enhance the ability to predict the distribution of algae in the environment with partial observations.

1.2 Problem statement

The environment is modeled as a grid map, represented by a connected graph $G = (V, E, \omega)$. Each node $v \in V$ corresponds to a location an ASV can reach and where it can take a measurement of the water parameters, while each edge $(u, v) \in E$ represents an unobstructed and shortest path between adjacent locations. The grid is 8-connected, it means that every agent can move up, down, left, rigth, up-right, up-left, down-right, down-left.

Time is discrete in this model, moving through every edge takes 1 temporal unit.

The $\omega : V \to \mathbb{R}_{\geq 0}$ is a function that assigns to each node a value which indicates how much algae is found in that particular area, this value is called: importance.

The $w(v, t) : V, T \to \mathbb{R}$ is a function that outputs, for every node and time step, the idleness value, which is basically the indicator of how much time has passed since the last measurement in the considered node. For every step in which the node v was not visited, a cost of τ is added to the idleness value of that node v. This is the formalization:

$$w(v, 0) = W_{MAX} \tag{1}$$

$$w(v, t+1) = \begin{cases} \min\{W_{MAX}, w(v, t) + \tau\} & \text{if } v \notin M(t). \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Where $M(t)$ are the nodes visited during the timestep t by the agents and $W_{MAX} = 1$.

An optimal solution to NHPP can be defined as a set of paths $\pi*$ that minimizes an environment-cumulative weighted idleness over the mission horizon T. That is,

$$\pi^* = \arg\min_{\pi} \sum_{v \in V} w(v, T)\omega(v) \tag{3}$$

The following subsections provide a brief description of the main components already present in the previous project.

1.2.1 DRL for ASV actions

Double Deep Q-learning (DDQN) is a deep learning method that allows agents to learn to estimate the reward of taking a determinate action in the environment given the state

where they are at. With the objective of using this model to maximize the reward of every action they take.

During training the network receives as input:

- **Navigation map**: the 2d matrix containing a 0 where there are obstacles and 1 where the positions are visitable.
- **Agents positions**: the 2d matrix where a 1 is the position of the agent and everything else is 0.
- **Information model**: the 2d matrix outputted by the VAE-UNet with the entire algae bloom distribution estimate.
- **Idleness map**: the 2d matrix updated during every movement of the agents following the w function described in section 1.2.

By using a dataset with the data described in the list and the corresponding rewards, it can learn an estimate of the reward for all the 8 actions given the input state.

The conclusions of article [11] assert that the use of DDQL in NHPP is capable of optimizing large state-action policies for the ASVs. In the studies of the project's paper it is possible to see how this newly introduced DRL method performs better than the other models.

ϵ-greedy exploratory policy

During training, the agents choose, with a probability of $(1-\epsilon)$, the action that maximizes the reward and, with a probability of ϵ, a random action. ϵ is annealed over the training time to shift from exploration to exploitation behaviour.

Prioritized Experience Replay

To enhance the learning a replay memory buffer is used. In this buffer, experiences collected by agents are saved. From this buffer random mini-batches are taken, this can prevent temporal correlations in the sequence of experiences. A prioritized experience replay is implemented to give priority to experiences based on the difference between the reward known by the model and the one known by the new experience, the higher the difference the higher the priority.

Prevention of ASVs collisions

The project makes use of an algorithm to make agents choose actions where other agents are surely not placed. The priority of choice is given to the agent that promises a higher reward, followed by those promising less. They can go only where other agents are not

already placed, considering their position and a reasonable safety distance radius around them.

Reward function

The reward function is used to calculate the reward returned by the environment for every action made by the ASV. The function is:

$$r(s^t, a_j^t) = \sum_{v \in N(v)} \frac{w(v,t) * \omega(v,t)}{\rho(v)} \tag{4}$$

- s^t : the agent's current state at time t.
- a_j^t : the $j-th$ ASV's action to be evaluated at time t.
- $N(v)$: the nodes covered by the measurement of the ASV.
- $w(v,t)$ and $\omega(v,t)$: functions described in 1.2. A t parameter is added to the importance function to express the measurement of importance taken in the node v at time t.
- $\rho(v)$: the overlapping function, this function returns the number of ASVs that are covering node v, the minimum value for this term will be 1.

The higher the importance and the idleness the higher the reward, the lower the overlapping ASVs the higher the reward.

ASVs will prioritize locations with high importance, high idleness, and low coverage by other agents. Once the high-importance locations are measured, their idleness is immediately reduced to 0. Other locations with low importance and high idleness will be visited, but only after the high-importance locations have been addressed.

1.2.2 Introduction to VAE-UNet in the project

As previously mentioned, the VAE-UNet model is used in the project to estimate the full distribution of algal proliferation using only the measurements taken by the ASVs.

In [12], a VAE network is introduced for image segmentation, translating visual inputs into semantic maps, while employing independent CNNs to process prior and posterior distributions. In [10], a VAE framework is developed for reconstructing partially occluded solar radiation images. The loss functions from [10] and the prior/posterior architecture from [12] are used in this VAE-UNet architecture with modifications to suit the specific application.

The model is composed of two main components: a Convolutional Neural Network (CNN) (a UNet in this case) and a Variational Auto-Encoder (VAE).

The use of CNN enables the network to learn high-level features from the entire input data. The advantage of CNN is that the layers are not fully connected but make use of groups of weights which keep the spacial dependence, this enables the network to understand more about features in the images, in this case in the input matrix. CNN makes use of kernels, which are groups of weights that are applied multiple times across the image to "scan" it. This means that the number of parameters for high number of input data can stay reasonably low, while in a normal fully connected network there would be a considerably higher number of parameters.

The network's input are the following:

- a preprocessed model of the importance Y^t constructed by a composition of the samples taken by the agents so far in the locations in which they were taken.
- a binary visit mask M^t, with those places that have been sampled represented by a 1, and places that haven't been sampled yet represented by a 0.

The Variational Autoencoder (VAE) is introduced to improve the prediction of the UNet model by learning a lower-dimensional latent space representation, which is a way to compress input data while still preserving its important features and essential information. This latent space helps the UNet to detect more complex patterns.

The architecture includes two distributions: the posterior, trained with both input data and ground truth, and the prior, trained only with input data. During training, both adjust their weights to minimize the loss, with the goal of having the prior mimic the posterior. In evaluation, only the prior is used, assuming it generates a latent space similar to the posterior's one.

To enhance even more the model, a pre-trained VGG-16 model is used. In this case this model is set only in evaluation mode to avoid the training of it. This is used to obtain the difference between real data and VAE-UNet output data.

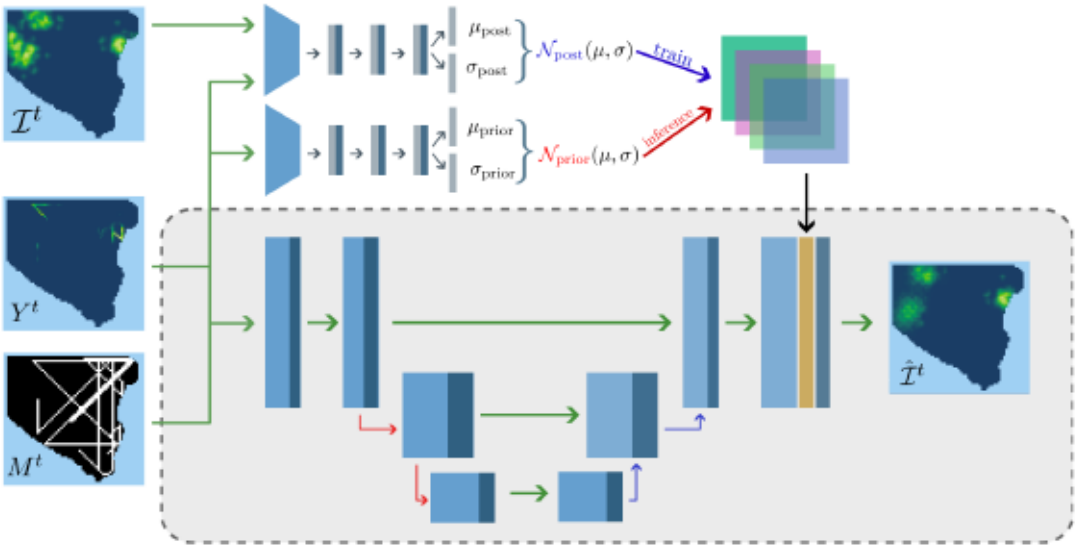


Figure 1: Architecture of the VAE-UNet visualized

The entire architecture is trained by minimizing the loss composed by a weighted sum of the following terms:

- **Reconstruction loss**: the Mean Squared Error (MSE) between the generated model and the real ground truth. This term is related to the capacity of the network to invert the observation and provide an estimate of the complete map.

$$\mathbb{L}_{recons} = MSE(\hat{\omega}, \omega) \tag{5}$$

- **Kullback-Leibler divergence**: The KL divergence (measures the difference between two distributions) between the prior and posterior distributions. By minimizing this term, the network learns the inner distribution of the data.

$$\mathbb{L}_{KL} = KL(N(\mu, \sigma), N(\hat{\mu}, \hat{\sigma})) \tag{6}$$

- **Perceptual loss**: The difference between the high-level feature map ξ of the output and the ground truth from a pre-trained deep convolutional model like VGG-16.

$$\mathbb{L}_{perceptual} = MSE(\xi(\hat{\omega}), \xi(\omega)) \tag{7}$$

The final loss is:

$$\mathbb{L} = \beta_0 \mathbb{L}_{recons} + \beta_1 \mathbb{L}_{KL} + \beta_2 \mathbb{L}_{perceptual} \tag{8}$$

The paper of the project, during testing and results, described this model as better performing than the more traditional methods mentioned in the state of art chapter 2.

1.3 Heterogeneous fleets with common objective

In the conclusions of the project, one area for future work is identified: The heterogeneous Fleet Patrolling with common objective. In this case the agents pursue a cooperative goal, but with different measurement abilities, different speeds and with different type of sensors like sonars, WQP sensors, or spectrographic cameras.

This future development has been embraced to experiment with the idea of adding a fleet of drones to assist the ASVs in monitoring areas that are usually less explored because the ASVs give them lower priority based on the reward function they use. The drones would help extend the system's ability to patrol these lower-priority regions, improving the overall effectiveness and coverage of the fleet. This paper is based on the development and research related to this heterogeneous fleet patrolling task. As already mentioned, since the VAE-UNet is crucial for providing the DRL network with an appropriate state for learning reward estimation, the first step in experimenting with this new idea is to focus on testing of the VAE-UNet itself. This work specifically addresses this need by implementing the drones in the simulator software, training the VAE-UNet with newly generated data, and testing the resulting models to determine whether the approach shows promising results for further development.

Chapter 2

State of the art

This section provides a concise overview of the methods traditionally employed to predict the distribution of algae blooms based exclusively on already made observations prior to the introduction of VAE-UNet. And methods to move agents in the environment prior to the introduction of DRL network.

2.1 Before VAE-UNet

2.1.1 Gaussian Process

Gaussian Process (GP) is a regression method, i.e. a method that, given some labeled data (input data and what is expected to obtain as an output) during training phase, models a distribution over possible functions that could fit the data. During evaluation phase, this learned function can be used to make predictions on unseen data, providing both a mean estimate and an uncertainty measure. If the data used is well chosen, the predictions are probabilistically consistent with the behavior observed during the training phase, producing responses in line with the training data.

The training of Gaussian process has a cubic complexity $O(N^3)$ with respect to the number of samples, this is not ideal for large datasets such as those involving patrolling paths in environments that can vary a lot in dimensions. Gaussian processes also have the problem of having a limited group of functions (kernel) that can be fitted to the data. Expanding the range of possible functions to fit more complex patterns would make the training process even slower.

In [13] the use of Bayesian optimization and Gaussian processes is proposed for optimal sampling with different agents. The search space for each vehicle is distributed using a Voronoi tessellation to have a smaller number of possible samples because it is known that having a high number of samples with this method would be a performance problem. In this case no cyclic criterion could be easily introduced in the work, something that in

this work is absolutely needed.

So Gaussian Processes struggle to work on large-datasets while, in this project, scalability and efficiency are primary goals to achieve. Deep learning however not only can learn a broader range of complex functions capturing more intricate patterns in data but it is also a lot more scalable having the ability to leverage powerful hardware such as GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units).

2.1.2 K - Nearest Neighbors

K-Nearest Neighbors (KNN) is another regression method. Labeled data is still given to this model and, during training phase, it just simply stores all the data points with the dimensions of the storage structure corresponding to the number of input variables. During evaluation we simply give a new unseen input, KNN looks for the K elements that are the nearest to the new point and it simply assigns to the new input the average of those nearest data points already known and stored in the model.

Even in this case, when having a large dataset, the model will not perform well. The more data points saved in the model, the more calculations need to be done, making the model extremely dataset size dependent, even in evaluation phase. KNN's approach to learning is also less sophisticated. Compared to deep learning models, simply using the average is a relatively simplistic method.

2.2 Before DRL

2.2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an algorithm that can be used for the task of finding a local or absolute minimum or maximum in an objective function. This method does not learn anything in the way that deep learning does. PSO makes particles, which could be the ASVs, move for a random and always decreasing distance following three directions:

- Inertia: the direction that the ASV followed the previous step.
- Personal best location: the stored location where the $n-th$ ASV found it's personal best function value.
- Global best location: the location where the best value among all the values observed by all the ASVs has been found so far.

It can be seen how the Particle Swarm Optimization (PSO) method is used to find the maximum contamination levels [14].

In [15], a modification of the classic PSO algorithm is proposed, it encourages exploration using a predictive uncertainty of GP as a surrogate model. The main problem with

applying these methods for patrolling is that they are not intended for continuous monitoring. As already said GP tend to get worse with the increase of the number of samples, which is not ideal for large amounts of information from patrolling paths.

Since PSO is only optimizing and not learning, with unseen inputs, unexpected behaviors could occur due to the lack of pattern recognition and generalization abilities.

Deep learning can capture more complex patterns and generalize more effectively to unseen data, making it a more suitable model for this scenario.

Chapter 3

Implementation of the new components

This chapter describes the idea behind the new task of the project and the details of its implementation.

3.1 Introduction to heterogeneous fleets

3.1.1 Water quality parameters

When dealing with algae blooms, it is needed to know the best water quality parameters (WQP) to track in order to obtain the best estimate of algae presence. In article [16] it is mentioned that the WQP that have higher correlation with cyanobacterial abundance are:

- Color
- Temperature
- Total suspended solids
- N-tot (total nitrogen)
- P-tot (total phosphorus)

Among all the WQPs described in the previous list, color is a parameter that has drawn attention. Color can be detected through a photograph, without the need to be in close contact with the water surface. From this idea, the concept of introducing two different types of agents into the environment was developed: ASVs and Unmanned Aerial Vehicle (UAVs, i.e., Drones).

3.1.2 Roles of the fleets

Autonomous Surface Vehicles

The ASVs retain the same role as in the previous project, measuring WQP such as P-tot, N-tot, temperature, and total suspended solids. They follow the same reward function as before, prioritizing exploration of the most critical areas while leaving less important ones for later.

Drones

The main issue with relying solely on ASVs in the environment is that if a new algae bloom peak emerges in an unexplored area, it won't be detected by the model until an ASV eventually arrives to take measurements. Here is when drones become useful. Their reward function is focused on scanning areas with high idleness, regardless of their predicted importance. In fact, they aim to explore regions deemed less important, to identify any new peaks that may develop there over time. The reward function for the drones is:

$$r(s^t, a_j^t) = \sum_{v \in N(v)} \frac{w(v,t)}{\rho_{\text{Drones}}(v) * \rho_{\text{ASVs}}(v)} \tag{9}$$

In this case the reward considers all the nodes covered by the drone $N(v)$. The higher the idleness the higher the reward (in this case the idleness is updated not only by the ASVs but also by the drones in the same way as ASVs do it as explained in 1.2.1). The lower the redundancy of the drones $\rho_{\text{Drones}}(v)$ and of the ASVs $\rho_{\text{ASVs}}(v)$ the higher the reward. It could also consider the importance, trying to go for places with low importance, but it may be useful to have the drones end up in places with high estimated importance, mostly if the number of ASVs is not enough to cover all the algae peaks simultaneously.

3.1.3 Cooperation between the fleets

Although different sensors are used, the readings from the two fleets can be considered on the same scale, reflecting the actual significance of data collected from the water's surface. As previously mentioned, the ASVs perform the same tasks as in the old project, operating independently of the drones. The drones, on the other hand, venture into new areas where neither they nor the ASVs have been before, ever, or in a very long time, capturing environmental snapshots from above. These snapshots serve as input for the VAE-UNet, which now estimates the importance of the data by combining inputs from both the ASVs and drones.

The drones contribute to predict the overall distribution within the VAE-UNet, focusing on areas with high idleness.

ASVs benefit from the drone actions indirectly, using the importance model generated by the VAE-UNet to guide their future movements.

The drones rely on the idleness information gathered, which is also influenced by the ASV movements.

It can be said that ASVs communicate to drones where not to go through changes in their idleness maps. As a result, drones explore areas of high idleness where ASVs are not operating, reporting their discoveries through the importance model.

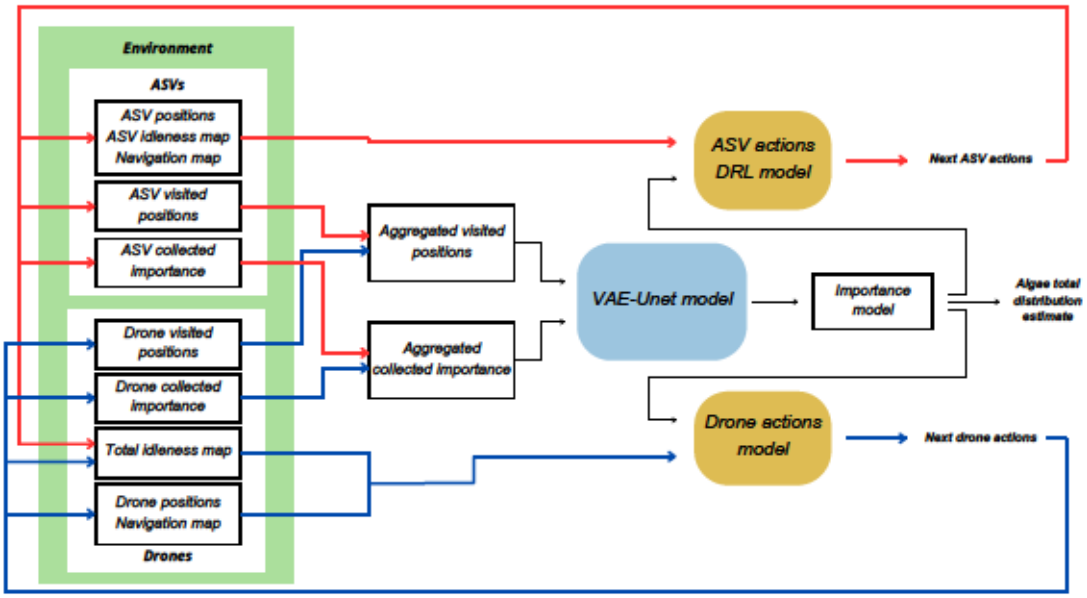
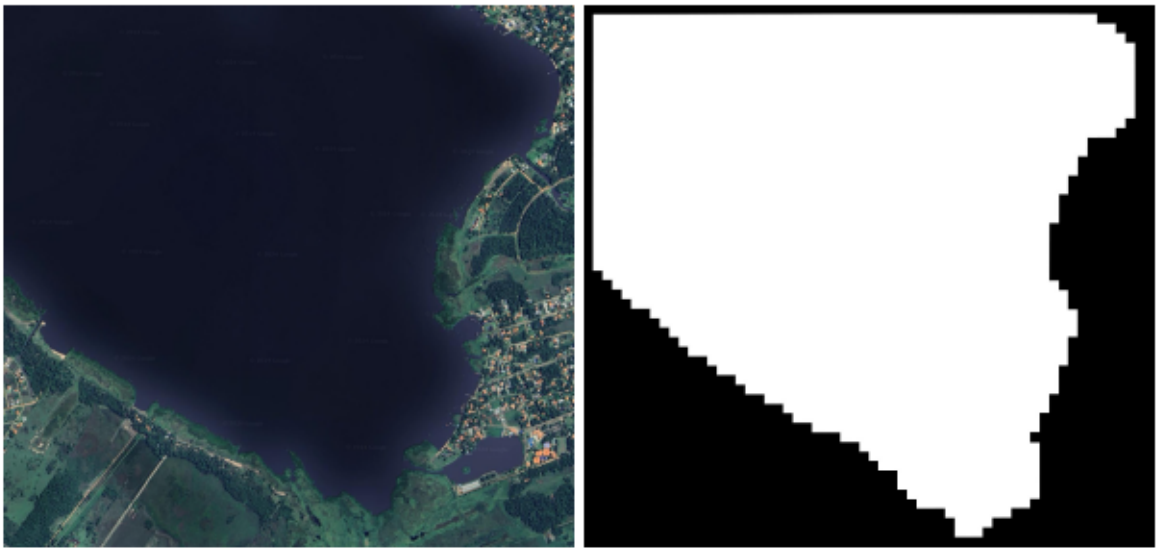


Figure 2: Data pipeline diagram after the introduction of the new drone fleet. The arrows indicating the data generated by ASVs are shown in red. The arrows indicating the data generated by drones are shown in blue.

As Fig. 2 shows, the first problem to solve for this task is the integration of the drones in the simulation environment and the testing of the VAE-UNet performances to see if there is some margin of upgrade compared to the old single fleet homogeneous model. This paper focuses closely on this aspect, which is why, in the diagram, a specific name has not yet been assigned to the drone's action decision model. However, it is assumed that, regardless of the model, it can still benefit from the same inputs that the ASV model already receives.

Throughout the work, as shown in Fig 3 the navigation map is consistently inspired by the southern part of Lake Ypacaraí in Paraguay (as in [6] [8] [13]), with 4 ASVs and 1

drone assigned for patrolling.



(a) Lake Ypacaraí (b) Navigation Map

Figure 3: The part of Lake Ypacaraí considered and its associated navigation map in the software environment

3.2 New components

This section is dedicated to the new software components implemented during this work. Before the introduction of the heterogeneous task the code already had these components:

- **PatrollingEnvironment**: it is the simulation environment, from where data can be extracted to generate datasets and where all the agents can be moved to see the effects on the environment by extracting observations and rewards.
- **Models**: these are the different models that make the prediction of the distribution of the algae on the whole water surface.
- **PathPlanners**: the component that decides the next action to take for the agents.

The new implemented components are described in the following subsections.

3.2.1 Drone

This class represents drones and it is introduced alongside the Vehicle class (that represents ASVs). The methods of the Drone class are:

- **constructor**: here are initialized different parameters, the new ones introduced for the drones are:
  - *influence_side*: the size of the side of the square that the drone covers with one reading, with the unit being the number of cells in the navigation map.
  - *camera_fov_angle*: the field of view (FOV) angle of the camera mounted on the drone.
  - *drone_height*: the height where the drone operates.

  For the whole time the influence_side is set to 9. To reach a value like this, considering the real width of the lake and the height of the drone fixed to 120 meters, the drone camera should have a 127° FOV.
- **reset**: method to reset parameters of the drone.
- **_influence_mask**: returns the matrix with the size of the navigation map, with a value of 1 only in the area that the drone covers considering the position where it is.
- **square_size**: utility method to calculate the square side in meters assuming that the ratio of the snap taken by the camera will always be 1:1.
- **move**: method to move the drone to the given position. If the new position is not part of the allowed positions in the navigation map it returns the string "COLLISION" and does nothing else.
- **collision**: method to check if the position given is part of the possible zones to visit.

The Vehicle class that represents the ASVs is pretty similar but, instead of having the influence_side parameter, it has the influence_radius parameter, which basically tells the radius of coverage of one read of an ASV.

3.2.2 CoordinatedHetFleet

This class extends the old CoordinatedFleet to represent the new fleet of drones and ASVs put together and handles the interaction of the two fleets with the environment. It makes use of the Vehicle and Drone classes to generate the fleet. The methods of the CoordinatedHetFleet class are:

- **constructor**: allows the call to old constructor and initialization of the new parameters introduced in this class:
  - Drone class parameters.

- *initial_air_positions*: the initial positions of the drones.
- *total_max_air_distance*: the maximum distance that a drone can travel before stopping to take measurements until the end of the simulation.
- *n_drones*: number of drones in the fleet.
- *update_only_with_ASV*: this flag allows to set if the increase of τ in the idleness has to be done only by the ASV and won't be done during the drone movements. If it is enabled the only thing that will be done during drones actions will be to set to zero the influence area of the drones.
- *influence_drone_visited_map*: this flag allows to set whether the visited map of the drones has to track only the effective position of the drones or to track the entire square of influence. In Fig. 4 it is shown the difference in the visited map between having the parameter set to True or False.

(a) influence_drone_visited_map = True (b) influence_drone_visited_map = False

Figure 4: Difference between the visited map with influence_drone_visited_map set to True or False

- *influence_ASV_visited_map*: this flag allows to set whether the visited map of the ASVs has to track only the effective position of the ASV or to track the entire influence that the ASV has (influence_radius).
- *check_max_distances*: flag to enable or disable the max_distances check on the distances traveled by the agents.

Other parameters that were already present in the old class, are for example the τ value to add for the idleness, the data for the ASVs which as said in the previous section 3.2.1 is almost completely similar to the Drone data.

- **move_ASVs**: method to move the ASVs and to edit the idleness map as said in the section 3.1.3.
- **move_Drones**: analogue method to move the Drones and to edit their total idleness collected as said in the section 3.1.3.
- **update_idleness_map**: used to update the ASVs idleness map.
- **update_idleness_map_drone**: used to update the total idleness map used for the drones.
- **get_ASV_position_map**: method to get the map with a 1 where the ASV given in input is placed.
- **get_drone_position_map**: method to get the map with a 1 where the drone given in input is placed.
- **get_ASV_fleet_position_map**: method to get the entire ASV fleet positions in the map.
- **get_drone_fleet_position_map**: method to get the entire drone fleet positions in the map.
- **get_ASV_positions**: method to get an array with all the positions of the ASVs.
- **get_drone_positions**: method to get an array with all the positions of the drones.
- **redundancy_ASV_mask**: method to get the redundancy mask which basically sums together all the single influence masks for the ASVs, as already said it is used in the reward calculation. The mask outputted is a matrix where the number in the cells indicates how many ASVs are measuring that particular area.
- **redundancy_drone_mask**: the same method as redundancy_ASV_mask but for drones.
- **changes_ASV_idleness**: method to calculate how much difference there was from the latest step to the current one in the ASV idleness map. The greater the change, the better the ASV's movement.
- **changes_drone_idleness**: same method but for drones idleness, which is the total one. As with the previous case, the greater the change, the better the drone's movement.

3.2.3 NoiseModels

With the use of drones some problems could arise. During the development of the task, the problems considered are: weather and hardware limitations. To reflect this potential problems in the simulation environment, three main noise models were implemented:

- **NoNoise**: In this case, the drone captures data that perfectly matches the ground truth. This is the best case scenario. It could be possible but with some expensive hardware and optimal weather conditions.

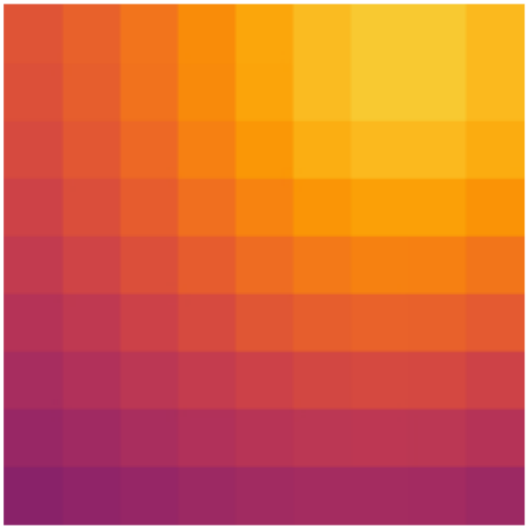
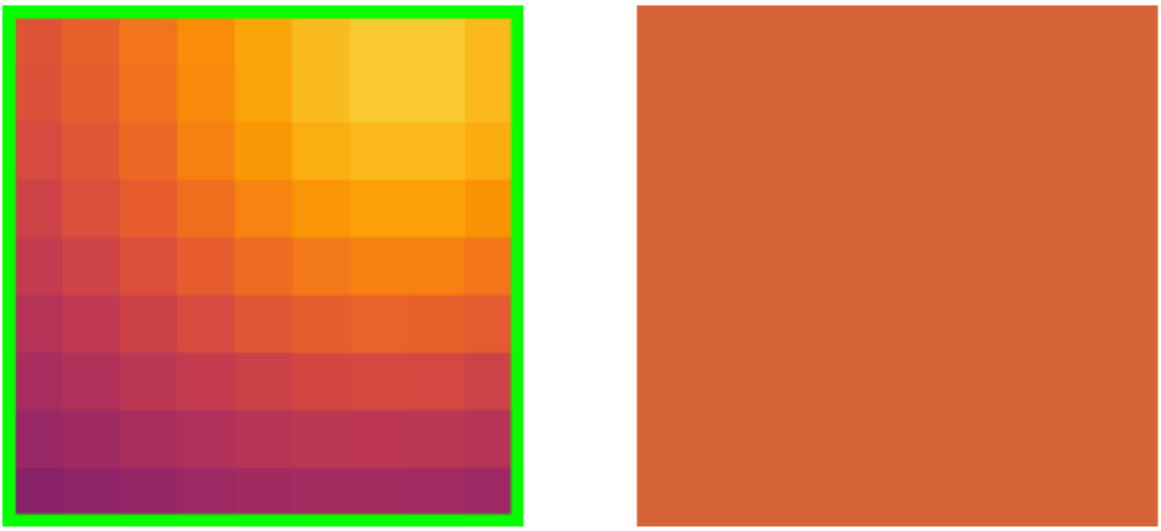


Figure 5: A drone measurement of the water surface with the NoNoise model

- **MeanNoise**: this noise model simulates the limitations imposed by fog or low clouds, which can cause blurriness in what the drone can capture. In this worst-case scenario, even if the color is detectable, all captured colors are averaged, resulting in a larger, uniform color square. Since the drone's height is much lower than the altitude of the clouds, this condition is practically impossible to occur. It is introduced mainly to evaluate the theoretical performance of the model under the worst-case noise model.

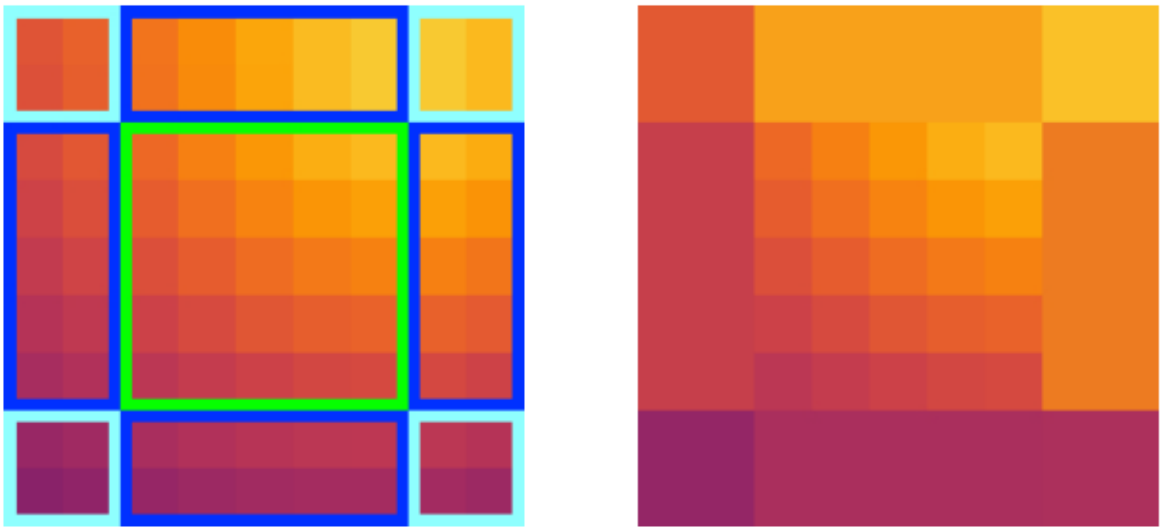


(a) Sections where the average is calculated (b) The output of the noise model

Figure 6: A drone measurement of the water surface with the MeanNoise model

- **FishEyeNoise**: This noise model offers a balance between the previous two models and simulates the capture from a high FOV camera. Such cameras distort reality more as objects or surfaces move towards the edges of the frame. By adjusting the drone's height, we can capture a central area with minimal distortion, while the outer regions become increasingly warped. To add a coherent noise, the model averages the color values extracted from the image in the external sections. As Fig. 7 shows, the model averages the color values from the four corner areas and the four side areas, but it leaves the central square, whose size is defined by the fisheye_side parameter, without averaging any value. This average is meaningful because the warped colors at the edges of the snapshot cannot be accurately placed in specific cells. The average indicates that, while the model is uncertain about exact placement, it recognizes that something in that color range exists in that position.

  For the whole time the fisheye_side is set to 5.



(a) Sections where the mean is calculated

(b) The output of the noise model

Figure 7: A drone measurement of the water surface with the FishEyeNoise model

3.2.4 HetModels

This components are made to store the observations of the importance done by the agents and output an estimate of the algae distribution. They have a common interface with these methods:

- **constructor**
- **update**: this method takes in input two flags that indicate what of the two types of agent made a measurement. It also takes as input the positions explored and the values of importance found. If the model is the VAE-UNet it also takes as input the visited map as already saw in section 1.2.2.
- **predict**: this method simply returns the estimate generated by the component.
- **reset**: entirely resets the importance map to 0.

The following are the two models that were implemented.

HetMiopicModel

This model is the most simple, it takes in input the measurements done and stores them in a common importance map that can be retrieved via the predict method. It is used to generate the datasets to train the VAE-UNet, since it simply stores the data gathered by the agents, which is one of the two inputs of the deep network. The fresher measurement is always, no matter what agent has done it, overwritten on old ones.

HetUNetModel

This model uses the previous one to store the importance measurements of the agents and requires another input which is the visited map, with this data it is possible to feed the trained VAE-UNet and retrieve the new generated prediction via the predict method.

3.2.5 DiscreteModelBasedHetPatrolling

This class extends DiscreteModelBasedPatrolling and serves as the high-level interface for interacting with external inputs, which represent the actions of the agents chosen at each step. Its primary role is to coordinate and execute internal changes based on these inputs, while providing a rendered dynamic environment and the ability to extract observations and rewards, primarily for dataset creation. The methods of the DiscreteModelBasedHetPatrolling class are:

- **constructor**: allows the call to old constructor and initialization of the new parameters introducted in this class:
  - Drone class parameters.
  - CoordinatedHetFleet class parameters.
  - *influence_radius*: the influence radius of ASV measurements.
  - *reward_type*: the name of the reward type for the ASVs, the only one implemented is the one that is explained in 1.2.1.
  - *reward_drone_type*: the name of the reward type for the drones, the only one implemented is the one explained in 3.1.2.
  - *benchmark*: the type of benchmark used to generate the ground truths, so the real distribution of the algae in the lake. For this paper shekel function was always used to generate benchmarks. Shekel benchmark function is used as it is a baseline benchmark used in many other previous works about water quality monitoring [13] [15]. This function models the WQP in a similar way to the data gathered in the Mar Menor Data Server [17], a particular case of contaminated water quality resource. The WQPs present a smooth distribution over the waters with peaks and valleys.
  - *model*: the name of the model that will give the estimate of the full distribution of the algae and will store all the importance measurements if needed, the different models are explained in 3.2.4.
  - *dynamic*: flag to enable or disable the step by step dynamic movements of the distribution of the peaks of algae. This method allows the ground truth to change with small variations of the positions of the peaks of algae in the ground truth.

  - *seed*: to initialize the randomness generation to keep it replicable.
  - *int_observation*: to change the format of the observations, if it is set to False the range will be [0,1] while if it is set to True the range will be [0,255] a UINT8 format useful for grayscale rapid conversion and also for discretization of the output.
  - *drone_noise*: the name of the noise model that is applied as a filter to the square measurements of the drones, different models were explained in 3.2.3.
  - fisheye_side: parameter already discussed in 3.2.3.

- **get_ASV_positions** and **get_drone_positions**: to get the positions of the agents.
- **get_ASV_positions_dict** and **get_drone_positions_dict**: to get the dictionary with the couple agent_id:position.
- **reset**: resets the internal state of the class.
- **_check_drone_presence**: checks if in the given position there is already another drone.
- **max_action_id_drone**: returns the number of possible positions to go for the drones.
- **action_to_position_drone**: convert the number of the action to the position encoded.
- **position_to_action_drone**: convert the position to the encoded action number.
- **get_feasible_positions**: returns the possible positions to go to for drones.
- **step**: this is the main method to use when interacting from outside of the class, it receives the actions that ASVs and Drones have to take and executes all the internal state changes outputting observations, rewards and other useful data.
- **_get_sample_drone_positions**: returns the dictionary with drone_id and corresponding entire square positions covered by that drone.
- **update_model**: makes the measurements for both ASVs and drones at the occurrence, it also applies the drone_noise filter to the values read and then saves them in the model.
- **get_observations**: returns the data structure with all the observations being: agent positions, drone positions, ASV idleness, total idleness and estimated total importance map.

- **get_ASV_rewards** and **get_drone_rewards**: to get the calculated reward for every ASV and drone agent. The way to calculate those rewards is already disclosed in the subsection 1.2.1 and subsubsection 3.1.2.
- **get_done**: to check what agents reached the maximum distance to cover if check_max_distances is enabled.
- **get_info**: to get useful metrics data calculated on idleness and importance.
- **render**: method to render all the subplots of the different views of the environment.

3.2.6 HetPathPlanners

As we discussed in subsection 3.2.5 the DiscreteModelBasedHetPatrolling requires in input the actions of the agents to move them. This is exactly what the HetPathPlanners do, choose the actions to execute. Only one path planner has been implemented and it is used for dataset generation. The RandomMover, this module is made by two classes which are:

- **RandomVehicleMover**: class that allows the extraction of a random action to perform for every ASV. The way it chooses its movements is similar to a lawnmower: the $n-th$ ASV keeps going in a particular direction until it finds a near obstacle (a shore) (it does not ever collide because of a parameter that determines the distance where to check for obstacles, which usually is higher that the movement length). After the ASV finds an obstacle it chooses a new direction that does not collide with anything and starts to follow it until another obstacle is found. This planner is called Non-Redundant Path Planner (NRPP), as explained in [7].
- **RandomDroneMover**: class that allows the extraction of a random position where the drone moves next. To adapt the same non-redundant path planner technique also in the case of drones.

This random path selection is used to reduce the bias of the Importance Model respect to the type of path planner used in the application.

3.2.7 TimedDiscreteModelBasedHetPatrolling

Introducing heterogeneous fleets brings on the table another problem, which is the timing of the fleets. While ASVs move all simultaneously and with the same timing, drones can travel distances that can vary every single action.

During the project the assumed speed of ASVs is 3 knots, i.e. ~6 km/h (as web page [18] says). The assumed speed of the drone is ~70 km/h. It is possible to say that drones not only travel varying distances each timestep, but they are also far quicker than the ASVs.

To address the time sequencing the TimedDiscreteModelBasedHetPatrolling class was developed. It is a wrapper class that encapsulates the simulation environment, i.e. the DiscreteModelBasedHetPatrolling class, and allows to automate the timing sequence of the movements of the fleets. In addition to the simulation class and the path planner classes, the speed_ratio is also required as input. This new parameter is calculated with this formula:

$$\text{speed\_ratio} = \frac{\text{drone\_speed}}{\text{ASV\_speed}} \tag{10}$$

So with the fixed values for ASV and drone the speed_ratio for this project is:

$$\text{speed\_ratio} = \frac{70\text{km/h}}{6\text{km/h}} = 11.67 \tag{11}$$

This means that ASVs have a speed of 1 while drones have a speed of 11.67. The wrapper algorithm in this class is shown in Algorithm 1.

Algorithm 1 Heterogeneous Fleet Timing Algorithm

```
speed_ratio ← 11.67
continue_simulation ← True
asv_actions ← pick_asv_actions()
drone_actions ← pick_drone_actions()
timetable["ASV", asv_actions] ← movement_length / 1
drone_distances ← calculate_distances(drone_actions)
for (drone_id, drone_action) in drone_actions do
    timetable[drone_id, drone_action] ←
      drone_distances[drone_id] / speed_ratio
end for
while continue_simulation do
    min ← pick_min_time(timetable)
    actions ← timetable.get_elements_equal_to(min)
    for (action, time) in timetable do
        timetable[action] ← timetable[action] - min
        if action in actions then
            to_do.append(action)
        end if
    end for
    for action in to_do do
        timetable.remove(action)
    end for
    observations ← environment.step(to_do)
    for agent_id in missing_agents(timetable) do
        if agent_id == "ASV" then
            action ← pick_asv_actions()
            timetable["ASV", action] ← movement_length / 1
        else
            action ← pick_drone_actions(agent_id)
            distance ← calculate_distances(action)
            timetable[agent_id, action] ← distance[agent_id] / speed_ratio
        end if
    end for
    continue_simulation ← environment.check_continue_simulation()
end while
```

This way the interaction with the environment does not always include every single agent but can include only a subgroup of agents. Every time a step occurs the agents that moved are the ones that contribute to the upgrade of the importance model. Tables 1, 2

and 3 offer a simple example of what happens during the use of this algorithm (always assuming speed_ratio = 11.67).

The algorithm picks the minimum time value in the table which is linked with the agent or the agents that come first in the time sequence. Once it picked the agents and their actions it feeds them to the environment gathering the returned observations. It also subtracts the time that passed by to all the agents. Before reiterating it extracts a new action for every agent that just interacted with the environment.

	Distance	Time
ASV	2	2
Drone_1	18.51	1.5861
Drone_2	26.53	2.2733

Table 1: Example timetable at beginning of iteration

	Distance	Time
ASV	2	0.4139
Drone_1	18.51	0
Drone_2	26.53	0.6872

Table 2: Drone_1 is the first agent to complete its action, the other agents get closer to the completion of their actions

	Distance	Time
ASV	2	0.4139
Drone_1	9.54	0.8174
Drone_2	26.53	0.6872

Table 3: New action with relative time to complete is assigned to Drone_1

Chapter 4

Experimental Evaluation

4.1 Protocol

As already disclosed, the paper is about the training of a VAE-UNet model that can estimate the full importance distribution of the algae, with the addition of drone contributes in the environment. The first thing to do is generate datasets to train the VAE-UNet. VAE-UNet requires two inputs: the importance map and the visited map. To generate it, as Fig. 2 in Section 3.1.3 shows, measurements done by the ASV and the drone and places visited by the agents are aggregated to obtain the importance map and the visited map. There are different NoiseModels to use during training and there are also other parameter values which are still undecided.

The protocol is the following:

- Generating the training datasets by extracting visited positions, measurements done and ground truth from the simulation environment for every step, using different combinations of parameters that decide how the agents generate and store all this data.
- Training all the models with their respective training dataset.
- Choosing the best models via testing with validation datasets.
- Verifying if test results are approximately equal to the validation results to confirm the model's performances.
- Testing the chosen models to evaluate their behaviours in different scenarios.

4.1.1 Parameter setting

The parameters to set in the simulation for the generation of the different datasets are:

- *influence_drone_visited_map*: in subsec 3.2.2 this parameter is already explained.
- *influence_ASV_visited_map*: in subsec 3.2.2 this parameter is already explained.
- *HetMode*: parameter that sets how the importance read by the agents has to be stored into the HetModel used by the simulation environment.
- *NoiseModel*: the NoiseModel to apply to the drone measurements in the dataset that the model has to learn from, those models are already described in subsec. 3.2.3.

The influence_drone_visited_map was studied in both the True and False cases to find out which models fit data the best way.

The influence_ASV_visited_map was always set to False.

For the generation of the datasets the HetModel that collects all the data as already explained in subsection 3.2.4 is the HetMiopicModel , it saves measurements only in the actual cells explored for the ASVs, without any influence_radius to consider. In contrast, for the drones, it always saves the entire square that has been scanned. This model has been coded with the name value of 'none' and this setting was always used in all the datasets.

The NoiseModels were studied in all the three possible settings.

As already said in 3.1.3 the simulation is done with 4 ASVs and 1 drone.

The following are the models trained:

Model_Name	NoiseModel	influence_drone_visited_map	With Drone
NoDrone	-	-	×
NoNoise	NoNoise	False	✓
	NoNoise	True	✓
MeanNoise	MeanNoise	False	✓
	MeanNoise	True	✓
FishEyeNoise	FishEyeNoise	False	✓
	FishEyeNoise	True	✓

Table 4: Models trained during the work

The first row references a model based on the old environment (without the drones) that was trained to make comparisons between the new heterogeneous fleet models and the old project's homogeneous fleet model.

The other rows reference the models based on the new environment with the drones.

4.1.2 Dataset creation

In this project the models were trained with static ground truth in mind. This means that for every simulation (ground truth) there is a number of steps during which the agents move in the environment and take measurements of the same fixed ground truth.

The number of steps to generate for every ground truth was set to 321 (from 0 to 320). With the new heterogeneous fleet and this number of steps, the surface of the lake in the simulation is covered by 90% after around 138 steps.

About 42% of the 321 steps in each generated simulation covers the moments when the lake is explored for less than 90%. The remaining 58% covers the moments when the system operates for most of the time, i.e., given the continuity that the patrolling task implies, with almost every point of the surface (more than 90%) analyzed at least once. The choice of 321 steps was done to allocate a solid 58% of the dataset for moments of high measurement density, which occur all the time after the initial steps.

To generate this datasets, data is extracted from TimedDiscreteModelBasedHetPatrolling simulation with RandomMover (section 3.2.6) as path planner. Every data point of the dataset as Fig. 8 shows is always composed by: visited map, importance map and ground truth.

Figure 8: Example of a data point of NoNoise training dataset with influence_drone_visited_map set to False. From left to right: visited map, importance map and ground truth

To train, validate and test the models, the datasets all have 321 steps for every ground truth. Training dataset is composed by 3000 ground truths, validation dataset is composed by 100 ground truths and testing dataset is composed by 50 ground truths.

In Fig. 9 the simulated lake is decomposed in the data format used in the code to better understand what goes on under the hood.

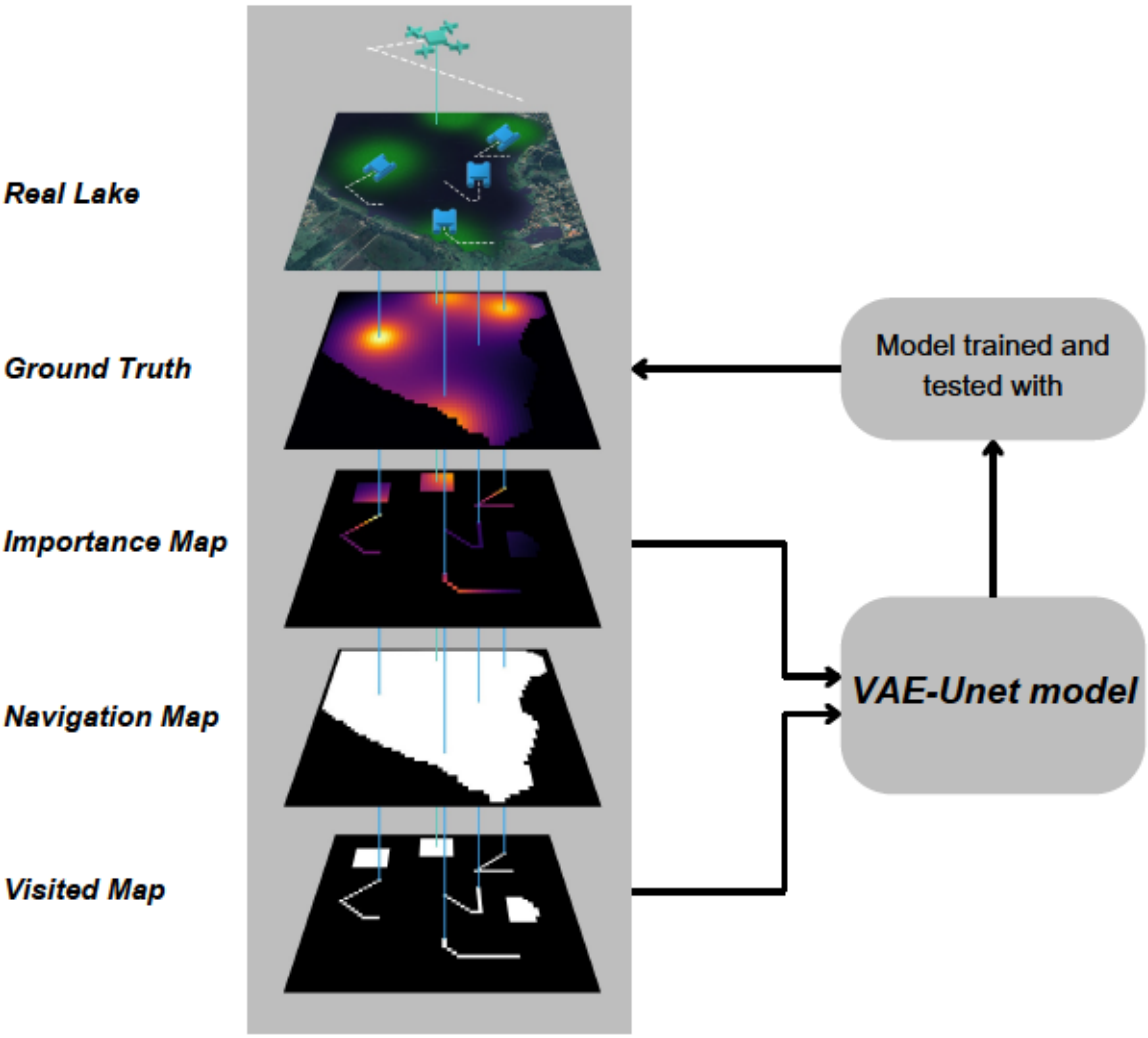


Figure 9: Decomposition of the simulation into the data used during VAE-UNet training and testing. In this case the drone and ASVs are working with the NoNoise model and with influence_drone_visited_map set to True

4.2 Training

During all the training sessions, a batch size of 64 and 50 epochs were consistently used. The training process chosen uses a method called checkpointing. This means that, whenever a new epoch results in a lower loss compared to the previously saved model, the model's weights are updated to reflect this new and improved version. Those weights are then used in the next phases.

When using an NVIDIA A100 Tensor Core, the total training time for each session was

around 25 hours. With an NVIDIA GTX 1070, the training took approximately 60 hours.

The following are the loss graphs of the models training sessions, the title of every graph contains the encoded name of the model trained. The name is composed with the values of the parameters discussed in subsection 4.1.1 in the following way:

$$NoiseModel\_ - \_ \tag{12}$$
$$influence\_drone\_visited\_map\_ \tag{13}$$
$$influence\_ASV\_visited\_map\_ \tag{14}$$
$$HetModel \tag{15}$$

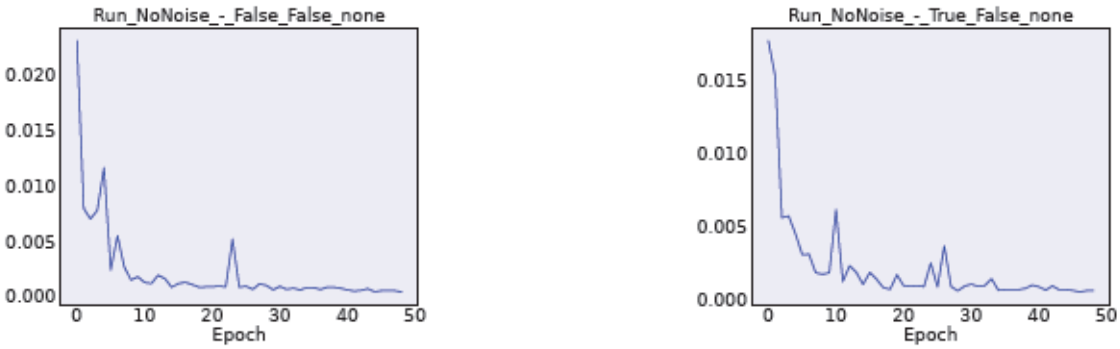


Figure 10: NoNoise models training loss

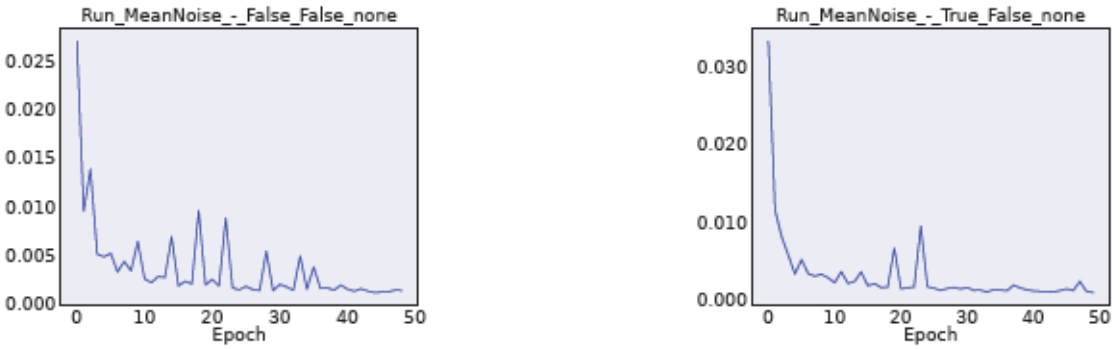


Figure 11: MeanNoise models training loss

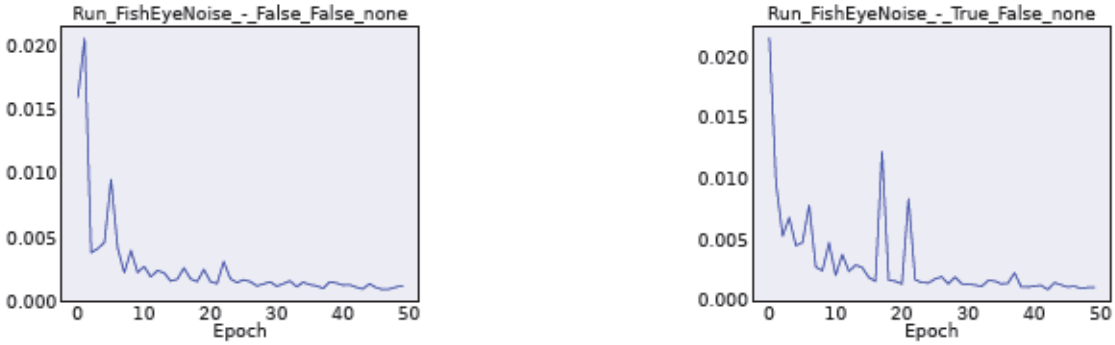


Figure 12: FishEyeNoise models training loss

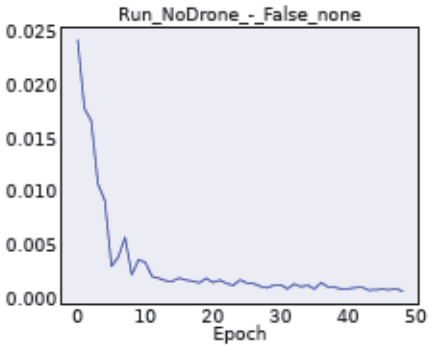


Figure 13: NoDrone model training loss

4.3 Validation

After training all 7 models, one model had to be selected for each pair that shared the same type of drone noise, so at the end of this selection a total of 4 models was obtained. The metrics used to determine which model is the best performing were:

- Mean Squared Error (MSE): it measures the average squared difference between the estimated model and the ground truth.
- Root Mean Squared Error (RMSE): square root of MSE, it measures the average residuals between the estimated model and the ground truth at each timestep.
- Weighted RMSE (W_RMSE): it adjusts the RMSE by giving more weight to high importance zones, so in higher-priority areas RMSE has a greater impact on the overall score.

The models that performed better on at least two out of those three metrics were chosen. These are the picked models from table 4 in subsection 4.1.1:

Model_Name	NoiseModel	influence_drone_visited_map	With Drone
NoDrone	-	-	×
NoNoise	NoNoise	False	✓
MeanNoise	MeanNoise	False	✓
FishEyeNoise	FishEyeNoise	True	✓

Table 5: List of new models to train

The performances tested on this selected models with the validation datasets are shown in tables 6, 7 and 8. The improvement is calculated in comparison to the old NoDrone model.

	Model_Name	AVG MSE	Improvement
1°	NoNoise	~ 0.000291	74%
2°	FishEyeNoise	~ 0.000389	65%
3°	NoDrone	~ 0.001125	-
4°	MeanNoise	~ 0.002168	-92%

Table 6: MSE based ranking

	Model_Name	AVG RMSE	Improvement
1°	NoNoise	~ 0.005088	62%
2°	FishEyeNoise	~ 0.009237	31%
3°	NoDrone	~ 0.013420	-
4°	MeanNoise	~ 0.037669	-180%

Table 7: RMSE based ranking

	Model_Name	AVG W-RMSE	Improvement
1°	NoNoise	~ 0.007939	63%
2°	FishEyeNoise	~ 0.015058	30%
3°	NoDrone	~ 0.021558	-
4°	MeanNoise	~ 0.063413	-194%

Table 8: W-RMSE based ranking

4.4 Testing

After the validation, the four selected models were tested to check the final performances and also to check if there were major differences between validation and testing results.

As it is possible to see in tables 9, 10 and 11 the difference between validation and testing results are practically non existent, the training of this models and the choices of the training dataset parameters were correct and the non different performances between validation and training shows it. NoNoise and FishEyeNoise appear to be notably better than the old NoDrone model. It is possible to see that MeanNoise, even with the addition of the drone agent, averagely does a lot worse in comparison with the old homogeneous fleet. In the next section 4.5, the models are tested more deeply and this last observation, along with other ones, are made.

	Model_Name	AVG MSE	Improvement
1°	NoNoise	~ 0.000313	76%
2°	FishEyeNoise	~ 0.000431	67%
3°	NoDrone	~ 0.001339	-
4°	MeanNoise	~ 0.002413	-80%

Table 9: MSE based ranking

	Model_Name	AVG RMSE	Improvement
1°	NoNoise	~ 0.004977	64%
2°	FishEyeNoise	~ 0.009038	35%
3°	NoDrone	~ 0.013909	-
4°	MeanNoise	~ 0.037542	-170%

Table 10: RMSE based ranking

	Model_Name	AVG W-RMSE	Improvement
1°	NoNoise	~ 0.007817	64%
2°	FishEyeNoise	~ 0.014543	34%
3°	NoDrone	~ 0.022151	-
4°	MeanNoise	~ 0.062037	-180%

Table 11: W-RMSE based ranking

4.5 Models put to test

In this section some tests done with the best validated and tested models are shown and explained.

4.5.1 Static test

In Fig. 14 there is a graph of the error between the prediction and the actual values for each step of the test dataset.

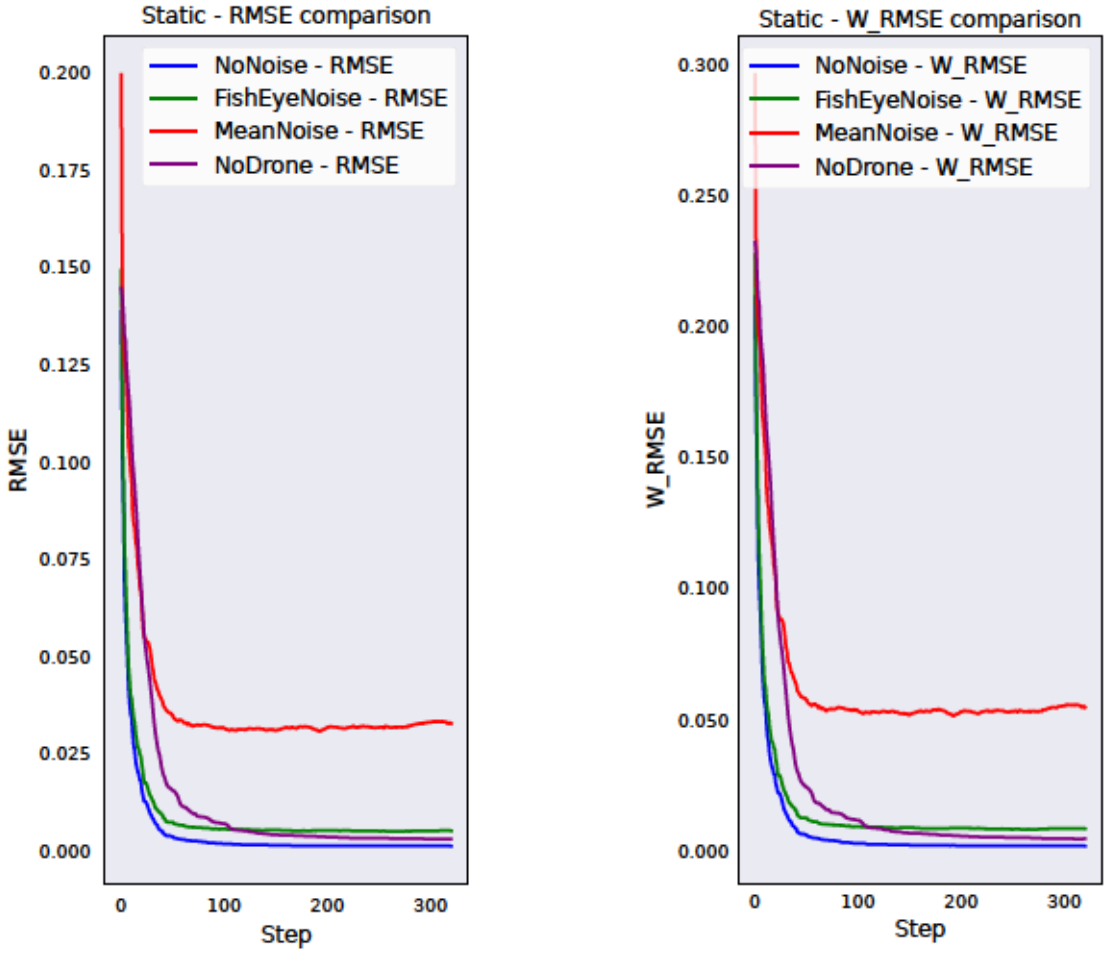


Figure 14: Graph for the static test, from left to right RMSE and W_RMSE graphs

It is possible to see that NoNoise is the best at finding the lowest error.

MeanNoise stops at a very high error. In this case, the drone measurements with this type of noise are more disturbing than helpful, leading the model to be unable to predict below a certain error threshold.

At the beginning of the execution, FishEyeNoise performs much better than NoDrone, thanks to the drone being able to map the surface much more quickly. Around step 120, however, the model with FishEyeNoise seems to start predicting with more error compared to the simpler homogeneous model. As already explained in section 4.1.2, after approximately the 138th step, the lake surface coverage from the drone readings begins

to exceed 90%, the shift between FishEyeNoise and NoDrone happens right around this moment. This could mean that after a certain level of coverage, as in the case of MeanNoise, even the FishEyeNoise drone readings begin to cause more interference than help. Since NoNoise consistently performs best, we can deduce that the noise introduced is the issue causing the drone models to start performing worse. Additionally, it can be noted that, in a static environment, using drones can be highly effective during the early stages of exploration. However, once sufficient coverage is achieved, unless drones with perfect reading conditions are available, it may be more practical to turn off the drones and switch to the NoDrone model.

4.5.2 Peaks variation test

One of the reasons why drones are introduced as a fleet is also to counteract the delay with which the old homogeneous model is able to discover new peaks or the disappearance of such. This test (graph in Fig. 15) is created with the intent of inserting or removing in a totally random manner a peak of algae presence in the ground truth. The dataset aims to evaluate how the models respond when both drone and no-drone scenarios encounter a sudden change in an area that was not visited for a while during previous steps.

The dataset used for this testing is composed of 20 simulations and 3001 steps for each simulation, every 300 steps a removal or addition of a new peak in the ground truth was randomly done.

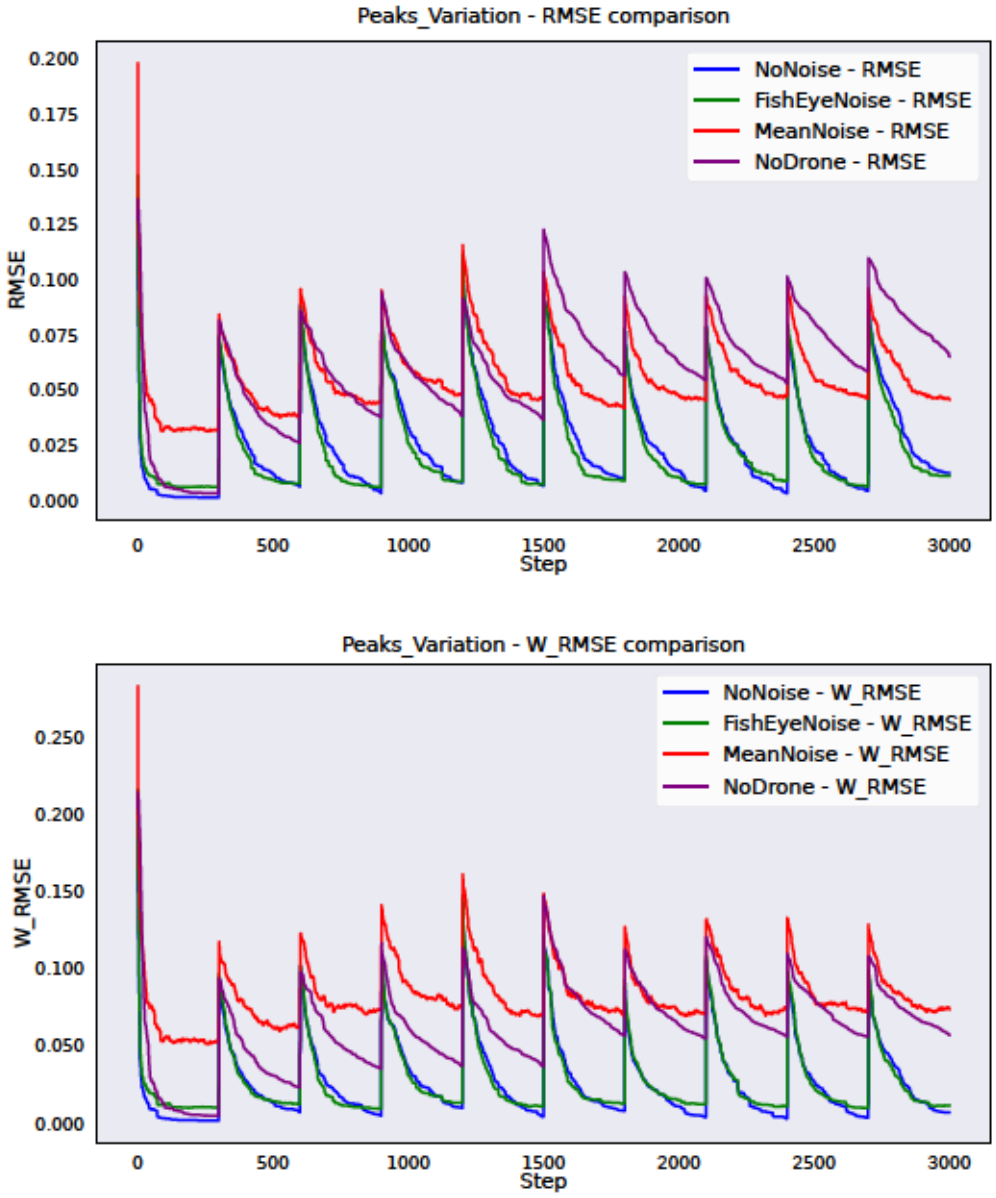


Figure 15: Graph for the peaks variation test, from top to bottom RMSE and W_RMSE graphs

Each peak observable in the graph is justifiable by the sudden change of the ground truth and therefore an inevitable increase of the error during first phases of new exploration. Looking at the weighted RMSE graph it is observable how NoDrone and MeanNoise are around the same level of error every time, with MeanNoise performing worse

before every peak variation. After some variations of the ground truth the difference between the NoDrone-MeanNoise couple and the FishEyeNoise-NoNoise couple gets even bigger, with the first couple's error getting always worse and the second couple's error remaining consistent at the same level.

The challenge between FishEyeNoise and NoNoise is interesting, as seen in the previous tests at the first descent NoNoise manages to predict reality better and immediately thanks to its drones without noise. In the subsequent phases, however, a particular behavior seems to continue. FishEyeNoise manages to reduce the prediction error faster than NoNoise and in the end both models reach around the same level of error. Interesting behavior since it differs from what was seen in the static test where NoNoise managed to always be slightly better. From this test, we can see how the NoNoise model operates under the assumption that what the drone sees is an accurate reflection of reality. It attempts to fit these observations into its predictions as definitive truth. However, after experiencing peak variations in the initial stages, the model can misinterpret outdated ground truth data, either lacking recent peaks or containing old ones, leading to a higher error perception. In contrast, the FishEyeNoise model has been trained to approach its observations with a degree of skepticism, thanks to the noise injected into its dataset. This training has enabled it to develop a more nuanced understanding of the various realities it encounters, allowing it to generate more realistic predictions right from the beginning. After some time, however, drones manage to scan enough surface to find new data and overwrite the old one, making NoNoise model reach the same performances or slightly better performances compared to FishEyeNoise model.

In an environment where these peak variations can occur and where timing is important. It is safe to say that a fleet of drones, even with controlled disturbance to the readings as those of FishEyeNoise model, can contribute very positively to the continuous patrolling of the water surface.

4.5.3 Dynamic test

The peaks variation test introduced some light dynamism to test a particular problem, namely the sudden discovery of significant variations in the ground truth. Now the same models (trained using a static environment) are tested with a completely dynamic dataset, that is, with a dataset that, at each new data point, slightly changes the position of the peaks of the algae distribution (thanks to the activation of the dynamic flag as explained in the subsection 3.2.5). This evaluation is mainly done to test the robustness of the new static models, and to see if, even in this case, they are better than the NoDrone and if some new interesting relationships are discovered between the various models with the drone agent. The graph in Fig. 16 represents the test done.

Also in this case the testing dataset is composed of 20 simulations and 3001 steps for each simulation, with the difference that the ground truth changes at each single step.

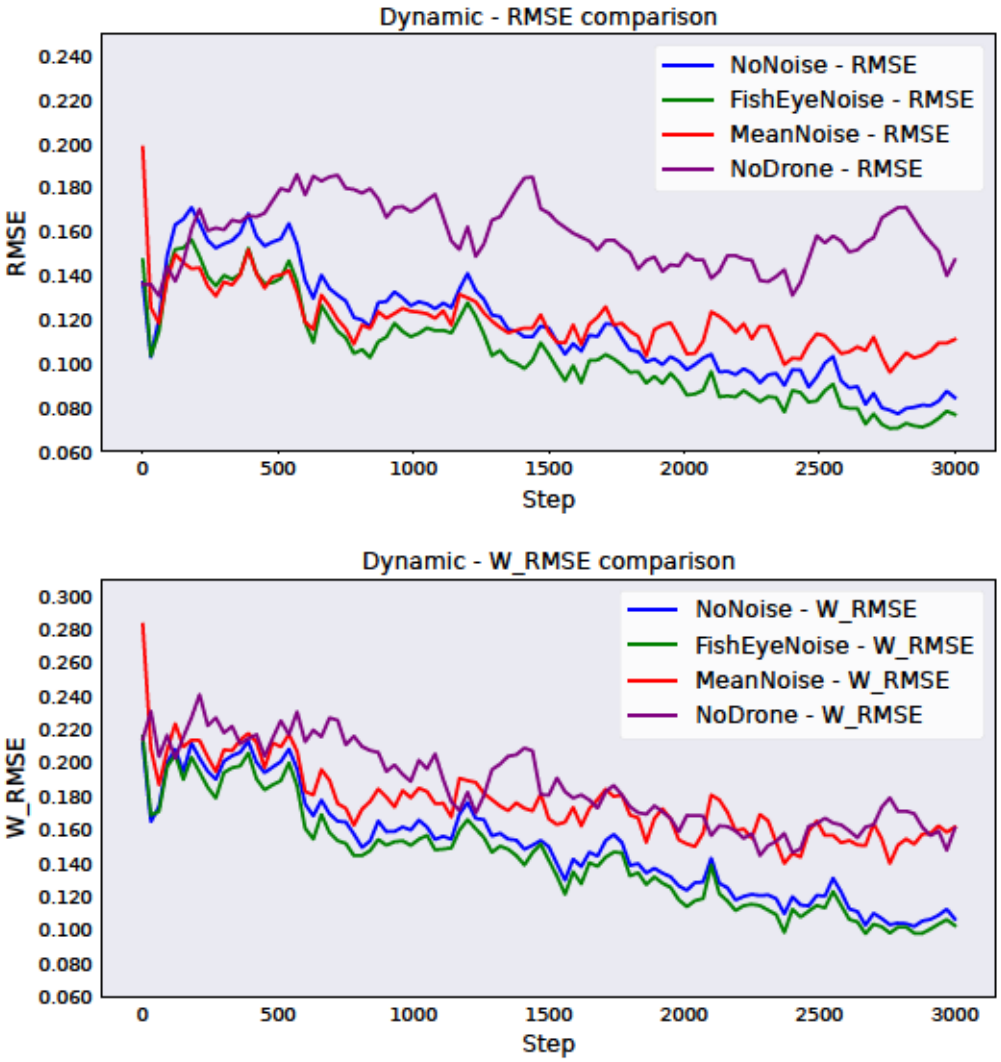


Figure 16: Graph for the dynamic test, from top to bottom RMSE and W_RMSE graphs. A moving average using a window of 30 values was applied to smooth the data, only the beginning value of the original graph was left unvaried.

The MeanNoise model as usual seems to follow the trend of the NoDrone error. What is still interesting now is how FishEyeNoise manages to match and even be slightly better than the NoNoise model, this confirms what already discussed in the previous test 4.5.2. With this test we can deduce that FishEyeNoise and NoNoise are the two models that have learned best from their dataset and in addition to performing better than NoDrone from all points of view, they also seem to react well to totally dynamic situations that they have never been able to observe during the training phase.

4.5.4 Dynamic Peaks Variation test

This test is done to put together all the dynamic features introduced in the two previous tests. The variation of the ground truth, by randomly adding or removing a random peak, introduced in 4.5.2 and the continuous change of the ground truth for every step of the simulation with a slight movement of every peak of the ground truth introduced in the subsection 4.5.3. Fig. 17 represents the test done.

Also in this case the testing dataset is composed of 20 simulations and 3001 steps for each simulation, the ground truth changes at each single step and the peak variation happens every 300 steps.

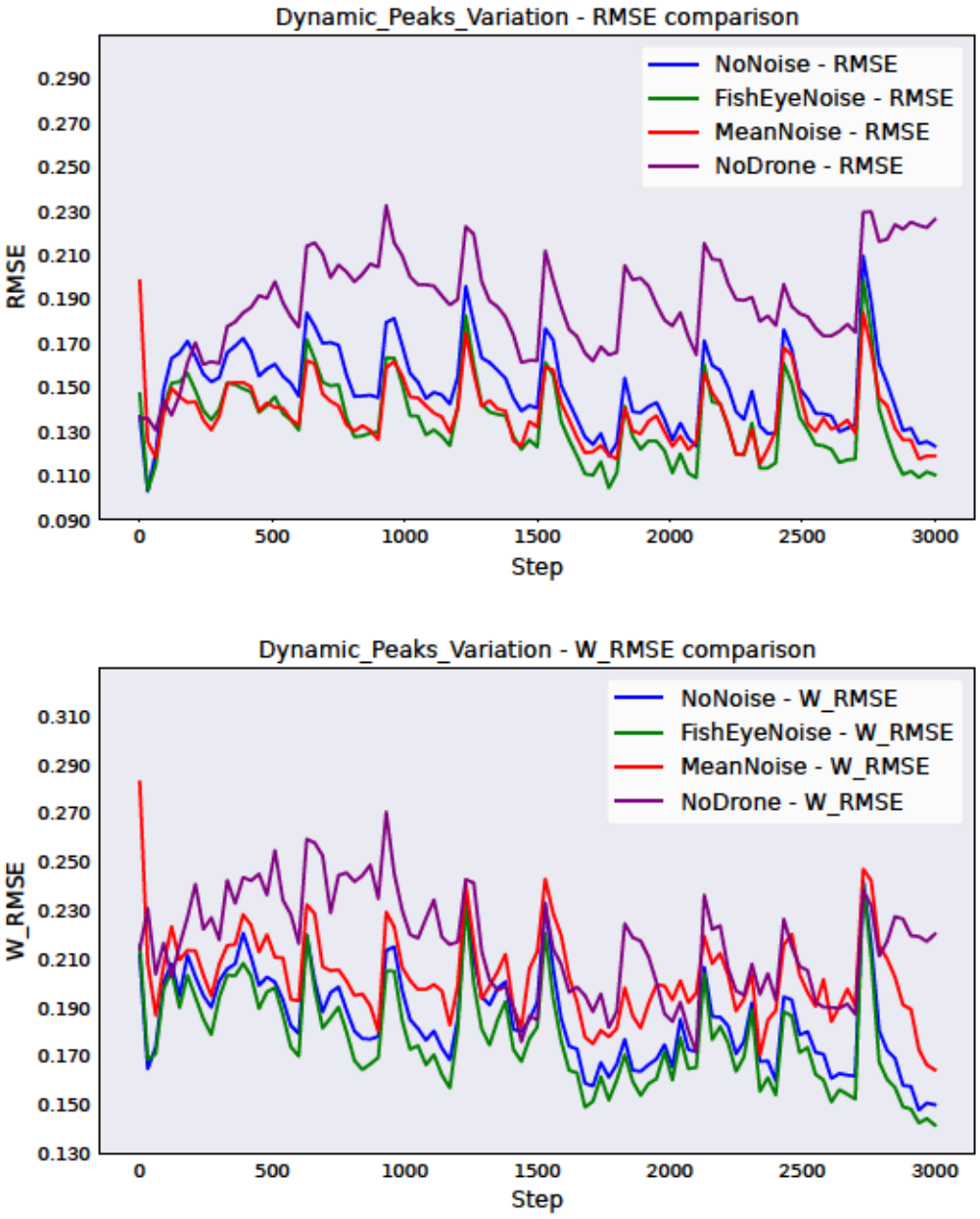


Figure 17: Graph for the dynamic peaks variation test, from top to bottom RMSE and W_RMSE graphs. A moving average using a window of 30 values was applied to smooth the data, only the beginning value of the original graph was left unvaried.

The peaks are justifiable because of the peak change every 300 steps. Looking at the weighted RMSE graph it is observable how NoDrone and MeanNoise also in this case

navigate on the same error level. The interesting fact that another time happens as in subsections 4.5.2 and 4.5.3 is that FishEyeNoise is slightly more robust to this dynamic inputs compared to the NoNoise model.

The fact that a model based on a heterogeneous fleet outperforms a homogeneous fleet model, even with some modest noise applied to drone measurements, makes this an intriguing area for further exploration and potential improvements.

Chapter 5

Conclusions

5.1 Conclusions

The goal of this project was to explore and analyze the performance of a new task related to the non-homogeneous patrolling problem, specifically through the introduction of drone agents to lower the estimate error of the distribution of the algae on the surface and to increase the speed of error drop, making the old project's homogeneous fleet a heterogeneous fleet with a common objective. The focus was on experimenting with the behavior of the VAE-UNet, a fundamental component whose output informs the action-selection of the various agents and generates the comprehensive distribution estimate of the algae on the surface. This work aimed to determine whether the addition of heterogeneous fleets could yield significant improvements to the estimation model.

Building on previous work, the project developed all the necessary components to add the drones in the simulation environment and to address all the new problems this integration yielded. Various models were then trained and tested to observe their behaviors in different scenarios and assess potential improvements over the old homogeneous fleet model.

As highlighted in Chapter 4, the findings indicate notable advancements in several areas, both in static and dynamic contexts (evaluated using models trained only with static datasets). This underscores the value of integrating drones as a new fleet, demonstrating their effectiveness in enhancing error drop timing and lowering error of the full algae distribution estimate. The work undertaken in this project contributes meaningfully to the understanding of heterogeneous fleet dynamics and opens avenues for future research.

5.2 Future developments

This work opens up several interesting areas for further development:

- One area is the implementation and analysis of a VAE-UNet with 4 inputs instead of 2, to distinguish between inputs from ASVs and drones rather than aggregating them. This approach could allow the network to discover more complex relationships by eliminating the overlaps that currently occur in both the map of visited positions and the importance map.
- Another key point to explore is the decision-making model for drone actions. Unlike ASVs, drones can move freely without distance constraints and follow a different reward structure. Since drones available actions are not always limited to 8, as with ASVs, their operational space is larger, though it still remains discrete (being made of all the position cells where they can go). Therefore, it becomes important to explore methods to make the decision-making model for drones scalable and still efficient, regardless of the size of the area being patrolled.

Bibliography

[1] Jill S. Baron, N. LeRoy Poff, Paul L. Angermeier, Clifford N. Dahm, Peter H. Gleick, Nelson G. Hairston Jr., Robert B. Jackson, Carol A. Johnston, Brian D. Richter, and Alan D. Steinman. Meeting ecological and societal needs for freshwater. *Ecological Applications*, 12(5):1247–1260, 2002.

[2] Weizhen Zhang, Jing Liu, Yunxing Xiao, Yumiao Zhang, Yangjinzhi Yu, Zheng Zheng, Yafeng Liu, and Qi Li. The impact of cyanobacteria blooms on the aquatic environment and human health. *Toxins (Basel)*, 14(10), September 2022.

[3] Mario Arzamendia Lopez, Daniel Gutiérrez, Sergio Toral, Derlis Gregor, Eleana Asimakopoulou, and Nik Bessis. Intelligent online learning strategy for an autonomous surface vehicle in lake environments using evolutionary computation. *IEEE Intelligent Transportation Systems Magazine*, PP, 01 2018.

[4] J. Sánchez-García, J.M. García-Campos, M. Arzamendia, D.G. Reina, S.L. Toral, and D. Gregor. A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications. *Computer Communications*, 119:43–65, 2018.

[5] Yann Chevaleyre. Theoretical analysis of the multi-agent patrolling problem. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, IAT '04, page 302–308, USA, 2004. IEEE Computer Society.

[6] Samuel Yanes Luis, Daniel Gutiérrez, and Sergio Toral. A deep reinforcement learning approach for the patrolling problem of water resources through autonomous surface vehicles: The ypacarai lake case. *IEEE Access*, 8, 11 2020.

[7] Samuel Yanes Luis, Daniel Gutiérrez-Reina, and Sergio Toral Marín. Censored deep reinforcement patrolling with information criterion for monitoring large water resources using autonomous surface vehicles. *Applied Soft Computing*, 132:109874, 2023.

[8] Samuel Yanes Luis, Daniel Gutiérrez Reina, and Sergio L. Toral Marín. A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The ypacaraí lake patrolling case. *IEEE Access*, 9:17084–17099, 2021.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[10] Kai Yi, Yi Guo, Yanan Fan, Jan Hamann, and Yu Guang Wang. Cosmovae: Variational autoencoder for cmb image inpainting, 2020.

[11] Samuel Yanes Luis, Daniel Gutiérrez-Reina, and Sergio Toral Marín. A dimensional comparison between evolutionary algorithm and deep reinforcement learning methodologies for autonomous surface vehicles with water quality sensors. *Sensors*, 21(8), 2021.

[12] Simon A. A. Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R. Ledsam, Klaus H. Maier-Hein, S. M. Ali Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images, 2019.

[13] Federico Peralta Samaniego, Daniel Gutiérrez Reina, Sergio L. Toral Marín, Mario Arzamendia, and Derlis O. Gregor. A bayesian optimization approach for water resources monitoring through an autonomous surface vehicle: The ypacarai lake case study. *IEEE Access*, 9:9163–9179, 2021.

[14] Micaela Jara Ten Kathen, Princy Johnson, Isabel Jurado Flores, and Daniel Guti errez Reina. Aquafel-pso: A monitoring system for water resources using autonomous surface vehicles based on multimodal pso and federated learning, 2022.

[15] Micaela Jara Ten Kathen, Isabel Jurado Flores, and Daniel Gutiérrez Reina. An informative path planner for a swarm of asvs based on an enhanced pso with gaussian surrogate model components intended for water monitoring applications. *Electronics*, 10(13), 2021.

[16] Marlena Piontek, Wanda Czyżewska, and Hanna Mazur-Marzec. Effects of harmful cyanobacteria on drinking water source quality and ecosystems. *Toxins (Basel)*, 15(12), December 2023.

[17] Marmenor. Sdc. `https://marmenor.upct.es/maps/Transparency`. Accessed: 2024-09-27.

[18] Autonomous Surface Vehicles | National Oceanography Centre. Autonomous surface vehicles. `https://noc.ac.uk/facilities/marine-autonomous-robotic-systems/asv`, Year. [Accessed: 19-Sep-2024].

Project developed at the Laboratorio di Sistemi Intelligenti Applicati (AIS Lab)
https://ais-lab.di.unimi.it/