IndoorExplorers: an OpenAI Gym environment for Multi-UAV Exploration Algorithms

Alexandra Fernandes
alexandra.fernandes@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2023

Abstract

The goal of this work was to create an OpenAI Gym environment, called IndoorExplorers, to simulate indoor exploration scenarios by a swarm of autonomous Unmanned Aerial Vehicles (UAV), each equipped with Light Detection And Ranging (LiDAR) sensors and with safe flying capabilities, including the detection and avoidance of any objects, across the space in question. The exploration tasks consists in determining the optimal path that gathers as much information about the space as possible, in this case to create a map of the space. Using a swarm of UAVs, it is possible to achieve these tasks faster, with fewer costs and safely for humans.

The developed OpenAI Gym-based environment was then used to test a Reinforcement Learning (RL) algorithm for path planning, specifically Dueling Double Deep Q-Learning (DDDQN). The developed environment currently allows tests in 2D maps with up to four UAVs equipped with a simplified simulated LiDAR sensor, with or without communications.

The results obtained compare two approaches to accelerate the training of the DDDQN. Furthermore, an analysis of the impact of more than one agent and whether communications affect the performance was done.

Keywords: Indoor Exploration, UAV Swarm, DRL, OpenAI Gym

1. Introduction

Unmanned Aerial Vehicles (UAVs), as the name suggests, are airborne devices that do not have any onboard crew or passengers, meaning that they can be controlled remotely or perform autonomous tasks.

Nowadays, these devices have a broad diversity of applications that range from security and surveillance [1], cinematographic filming [12], disaster management (which includes search and rescue operations or response to natural disasters)[2], monitoring of specific areas or infrastructure (such as borders, power lines, dams, etc.)[4], agriculture[17], to name a few.

The increasing demand of UAV technology is due to their potential to be faster, more efficient and safer than using human labour and putting human lives at risk. They can vary in weight, range, size, configuration, payload, engine type and performance characteristics. According to these attributes, it is possible to customise what sensors, cameras, communication protocol and navigation methods to use according to the application in mind.[14]

UAVs can work as a single unit or in coordinated groups, such as swarms. The latter option presents benefits such as dividing the workload among all units, being faster and able to accomplish larger and more complex tasks. An UAV swarm consists of a coordinated group of entities that communicate with one another, sharing crucial information and working together towards a common goal.

As previously mentioned, the context of this work is focused on a specific task, which is the exploration task. It consists in gathering as much information as possible in an optimal way. Concurrently, it is possible to combine this with a mapping task when it comes to unknown environments. Autonomous exploration grants the ability to achieve tasks in a more efficient way and without risking human lives in hazardous environments and in emergency situations.

In most exploration scenarios, a group of UAV reveals to perform better than a single UAV [8]. As explained above, a swarm of UAV is capable of jointly and efficiently performing the tasks mentioned above. In addition, they can cover larger and more complex spaces faster than a single UAV.

Having this context in mind, a suited simula-

tion environment was necessary to simulate the intended behaviour. Given the emergence of the topic, there are a few existing solutions but each with its own specifications, such as having integration with a standardised Reinforcement Learning (RL) platform such as OpenAI Gym, being suited for single-agent or multi-agent training, scalability, compatibility and so on.

In the midst of diversity there was no standardised simple solution that could accommodate all the features intended, specifically a solution that allowed multi-agent Deep Reinforcement Learning (DRL) training, with communications in GNSS-denied unknown environments, which is the goal of this project.

2. Background

Firstly, the work of Seel *et al.*[19] was the closest to what was intended in this thesis and was used as a main reference to build the developed environment, since the main goal was also to explore unknown GNSS-denied indoor environments, relying on Inertial Measurement Unit(IMU) and LiDAR data. In [19], a LiDAR framework has been developed to enable a UAV to autonomously navigate and explore unknown factory environments, with the intent of mapping it. This framework is designed to meet specific requirements, such as operating in areas GNSS-denied areas and relying solely on onboard sensors. Data from IMU and LiDAR sensors is used for independent decision-making and to create a digital replica of the factory. The system has been implemented in ROS Gazebo, and various training and testing scenarios have been created for evaluation. Unfortunately, no source code was available for the ROS Gazebo models and testing environments that were developed. On another hand, a clear idea of which algorithm could be used was mentioned: DDDQN, which was identified as the best performing DRL algorithm regarding 3D UAV navigation by [20], being an improvement of previous versions, namely Dueling Deep Q-Network [23] and Double Deep Q-Network [22], and Deep Q-Network (DQN) itself.

Having this work in mind, the initial intended architecture was something similar to Figure 1. This architecture included every necessary component for a complete simulation setup, namely:

1. **Physics simulator and Rendering** - these are interlinked and would be used to visualise the correct operation of the proposed algorithm and attempt to get the most accurate representation of the real word - concretely resorting to Gazebo;
2. **Robotics framework** - to program each UAV's operating routine - using ROS;
3. **Autopilot** - attached to the previous framework an autopilot would be required, being the bridge between robotics routines and control of the UAV model - concretely using PX4 Autopilot.
4. **AI framework** - to test the RL algorithms - developing with OpenAI Gym.

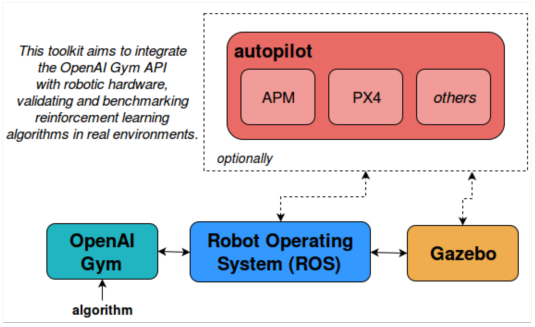


Figure 1: Simplified software architecture used in OpenAI Gym for robotics, from [24]

Given several compatibility issues with existing solutions, regarding versioning of different frameworks, where some used outdated or deprecated packages or software were no longer compatible with recent ones, and the lack of one centralised solution, the decision to create a simpler solution only based on OpenAI Gym was made.

The two following projects based on OpenAI Gym were used as primary foundations for the developed environment.

Firstly, [10] is a study that aims to connect advanced DRL techniques with the challenge of exploring and covering unknown terrains. To achieve this the MarsExplorer environment was created, designed specifically for exploring unknown areas. It is a OpenAI Gym environment that transforms the original robotics problem into a RL scenario that can be addressed by various readily available algorithms. Any learned strategy can be directly applied to a robotic platform without the need for a complex simulation model of the robot's dynamics, simplifying the learning and adaptation process. It is based on a grid like world, where a single agent equipped with a simulated LiDAR explores the unknown area. MarsExplorer was used as a foundation to develop this thesis, taking most of its features it was possible to create IndoorExplorers (the name was chosen accordingly to its inspiration). Additionally, four different state-of-the-art RL algorithms at the time were trained on the MarsExplorer environment, namely Asynchronous Advantage Actor Critic(A3C)[13], Proximal Policy Optimization(PPO)[18], Rainbow [7] and Soft Actor-Critic(SAC)[6]. Their performance was compared to human-level performance. Given the performance of PPO, its performance was then compared to a frontier-based approach. Demonstrating that the policy based on PPO could effi-

ciently adapt to unknown terrain while ensuring the coverage of areas that are costly to revisit, highlighting the effectiveness of RL-based approaches in exploration tasks.

Secondly, ma-gym[9] which is a collection of multi-agent environments based on OpenAI Gym (even though it is not on the official websites), IndoorExplorers was also based on the Predator-Prey environment of this collection. Predator-prey involves a grid world, in which multiple predators attempt to capture randomly slow moving prey. Each predator has a view mask corresponding to their cardinal direction, which means a prey is caught if it is within the field of view of at least one predator. This concept is similar to having a LiDAR sensor and for this reason it was also used as one of the foundations of this work.

3. Methodology

Before describing the environment, a set of assumptions were made in order to simplify the problem and are listed bellow:

- The area to be explored is considered to be a discrete space, divided into cells, whose dimensions are known (its height and width);
- Each agent's position is known at all times;
- Communication delay, including transmission delay, is negligible, but the communication system may have a limited range;
- Agents perform their actions sequentially, even though the choice of action is done previously. The order in which agents take their actions is randomized, so there is no priority amongst agents;
- The maps of agents in communication range are simultaneously and instantaneously merged and then a copy is saved for each agent;
- There are no collisions amongst agents - this is coded to be impossible;
- Each agent corresponds to one UAV flying at a fixed height.

IndoorExplorers based on the previously mentioned single-agent MarsExplorer environment, can have up to four agents, each one has a designated colour, agent 1 is blue, agent 2 is red, agent 3 is green and finally agent 4 is yellow. The LiDAR emulation was used unchanged from the MarsExplorer environment, but the rendering was entirely changed and is based on the ma-gym Predator-Prey environment (both were mentioned in previous section).

As can be seen in Figure 2, the LiDAR field of view can be visualised by the light blue circle that surrounds each agent, the radius corresponds to the pre defined communication range.

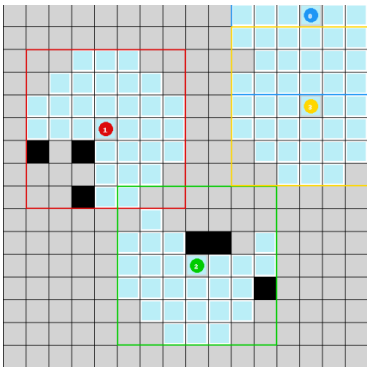


Figure 2: IndoorExplorers environment with 4 agents

Using OpenAI gym's simple configuration it is necessary to define four functions, which will be further described:

- step(action) - this function runs one timestep of the environment's dynamics, it is necessary to establish the core mechanisms for the environment. It receives an action that will be made effective inside this function and returns the information about the state, namely the observation, reward, terminated flag and some extra information.
- reset() - once the end of the episode is reached, this function must be called, which resets the environment to start a new episode. It only returns the initial observation.
- render() - function responsible for rendering.
- close() - usually closes the rendering window.

It is also necessary to have the algorithm to choose which actions to take, in this case it would be the DDDQN that will be addressed briefly. Additionally, the observation space and the action space must be defined, these specify whether the action space is discrete or continuous and what are their ranges. In this case, the MultiAgentObservationSpace and MultiAgentActionSpace from ma_gym were used. The observation space consists of a matrix that represents the map of what the agent sees, with float values ranging from 0,0 to the number of agents in the environment - each agent has a matrix called exploredMap which saves the updated real-time view of that agent. In Table 1 are compiled the meaning of each value of the matrix corresponding to each agent's map.

Values	Meaning
0.0	Unexplored cell
0.3	Empty/explored cell
0.5	Obstacle
≥ 1	Id of corresponding agent occupying that cell

Table 1: Meaning of values in each agent's map

With the MultiAgentObservationSpace wrapper, the observation space is defined as a list with the maps of all the agents, with the indices matching each agent's id minus one, meaning that the observation with index zero corresponds to the map of agent number one.

As for the action space, for a single agent it takes one of four possible discrete actions depicted in Table 2, there is one additional action for the multi-agent case, that is no-op standing for no-operation to allow agents to not move in case it is more favourable. Using the MultiAgentActionSpace, the action space is the same but once again it becomes a list with the length of the number of agents, where each value is the action assigned to the corresponding agent.

Values	Corresponding action
0	Moving down
1	Moving left
2	Moving up
3	Moving right
4	No-op

Table 2: Meaning of each action

In Figure 3, it is possible to see a flow chart that depicts the typical sequence of the previously described functions in a OpenAI gym environment for a single-agent scenario. Its simplicity is one of the reasons for its popularity. It consists of a loop with the desired number of episodes, that encloses a chain of choosing actions and then applying them in the step function. The step function then retrieves an *observation*, *reward*, the flag to whether the agent has entered a terminal state (*done*) and some extra information stored in *info*. When the done flag is activated the episode ends, triggering the reset of the environment, starting a new episode. The rendering is optional, but in this case it is done in the beginning of the loop. Once all episodes have been completed, the environment is closed.

In a multi-agent case, the structure is maintained, but each of the fundamental functions returns a list of variables, instead of a single reward the step function returns a list with the rewards for each agent, the same for the observations, and so forth.

Since the structure for the multi-agent code is based on the flowchart present in Figure 3, the actions for all agents are chosen first and stored in *action_n* array and this array is then used as an argument for the step function.

Inside the step function the order in which agents take action is randomised, thus avoiding any priority among agents. Before making the action effective, a verification is done to check if the move is

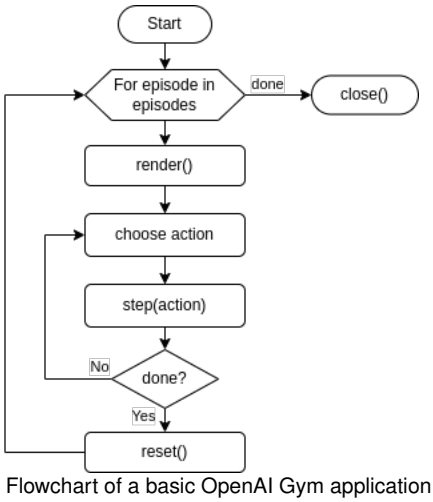
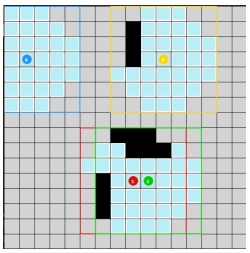


Figure 3: Flowchart of a basic OpenAI Gym application

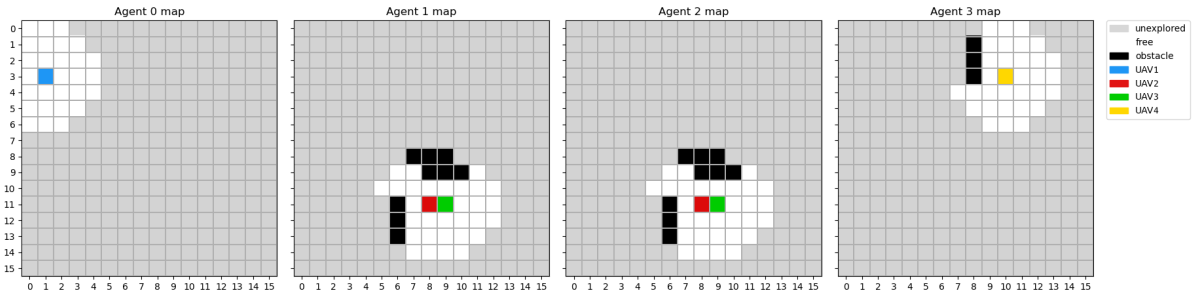
valid, a collision with an obstacle has happened, if it takes the agent outside the maps' bounds, or if the agent gets stuck between positions. If the next move would result in a collision with another agent, then it is cancelled and the agent does not move - this decision was made expecting it would result in faster learning. Regarding the getting stuck verification methods, this was implemented since a tendency for an agent to be in a cyclic path occurred, this will be explained further in the next section. This verification is carried out by checking the connected components in an indirect graph where each node is an agent, using the auxiliary adjacency matrix called *comm_range* and the Depth-First Search (DFS) algorithm (the idea for this comes from [5]). Having the connected components, each agent's explored map (corresponds to the updated map of what it sees) that is in range is then merged and a copy is saved in each, replacing the old explored map. In Figure 4 it is possible to observe one moment where agent 1 (red) and 2 (green) are in range of each other.

After merging the maps, if possible, the reward is calculated, and a verification on whether each agent has reached a terminal state is done. The episode only ends when all agents have reached a terminal state - this includes one of the following options:

- reaching the maximum number of steps (defined as 400);
- a collision with a wall or obstacle has happened;
- the agent goes outside the bounds of the map;
- a cyclic path has been found or in simpler terms the condition to be considered stuck has been reached;
- the map has been explored by the pre defined percentage (in this case 90%).



(a) Full view



(b) Each agent's explored map

Figure 4: Merging of maps demo

The reward system is explained in the next section, since several values where experimented. The general idea is that positive outcomes, such as successfully exploring the defined percentage of the map (in this case 90%), the agent is rewarded with a large positive value, whereas in the other cases (except the reaching the maximum number of steps) which correspond to harmful or disadvantageous situations, the agent is penalised with a large negative reward. Regarding each new cell that is explored, a pre-defined value is added for each - a copy of the previously explored map is saved and then compared with the current map, by subtracting the number of non-zero cells, it is possible to get the number of newly explored cells.

Advancing onto the algorithm, as stated the choice was to use DDDQN based on the work of [19]. In order to add this algorithm to the environment, an implementation from Robbie Estes was used as a foundation (the official github repository can be found in [3]) , who trained the network to play Joust, Ms. Pac-Man , Super Mario and Space Invaders, using gym retro [15] - which takes classic video game roms into OpenAI Gym environments for RL training of fully capable player agents and comes with integration for over a thousand games, from different consoles such as Nintendo, Atari, NEC and Sega.

The architecture of a DDDQN can be seen in Figure 5 (this image is based on the image in [21]). The network was prepared to have as input a stack of image frames of the game being played and outputting a vector of Q-values for each action possible in the given state, taking the biggest Q-value of this vector will give the best action for that state.

At the time, the dimensions of the network were adapted to receive a matrix which corresponds to the observation of the agent being trained. Initially the matrix was converted to an image represented by three matrices for the RGB values of the corresponding image and these were stacked, but the agent was not learning as intended since the input space was to sparse. At the present time, the realisation that a misunderstanding happened during this step, since this work was not a video game, then there was no need to use pre-processing of the video game frames and a few steps were removed - the map was already represented by a single matrix with well defined values. The belief that one crucial step was removed, namely, the stacking of frames which gives the network a sense of movement. This means the results were obtained using a single map, which can be considered as a single frame instead of a stack of frames - this will be further commented in the next section.

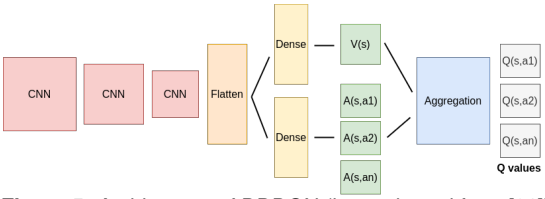


Figure 5: Architecture of DDDQN (image based from [21])

Proceeding to a brief description of the network, the first three layers are used to process the input frames to extract features. After flattening, the information stream is divided into two components.

To explain this, a reminder that Q-values, Q(s,a), correspond to how good it is to be at a given state *s* and taking an action *a* at that given state. This means Q(s,a) can be decomposed as the sum of V(s) - the value of being at that state *s* - and A(s,a) - the advantage of taking action *a* at that state *s* (how much better is to take this action versus all other possible actions at that state) - as can be seen in equation 1.

$$Q(s,a) = A(s,a) + V(s) \tag{1}$$

Concerning the aggregation layer, to generate the Q values for each action at that state, it is necessary to subtract the average advantage of all actions possible of the state, as evidenced in equation 2, in order to avoid the issue of identifiability in back propagation - not being able to identify A(s,a) and V(s), given a certain Q(s,a).

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{A}\sum_{a'} A(s,a') \tag{2}$$

Finally as explained before DDDQN takes advantage of the features of the Double DQN in order to avoid overestimation of Q values. The accuracy of Q values depend on what actions have been tried and what neighbouring states have been explored. As a consequence, at the beginning of the training there is not enough information about the best action to take. Therefore, taking the maximum Q value (which is noisy) as the best action to take can lead to false positives. If non-optimal actions are regularly given a higher Q value than the optimal best action, then the learning will be unstable. The solution is to use two networks to decouple the action selection from the target Q value generation. This is how the Double DQN helps to reduce the overestimation of Q values and, as a consequence, helps training faster and have more stable learning. The weights from the "q_eval" network (which is the name of the model in the code) are copied to the "q_target" model after *update_every* number of episodes.

To summarise, at present time, the main features of this environment are stated as follows:

- It can perform simulations with up to four agents;
- Each agent has a circular field of view with a pre-defined range, constrained by an emulated LiDAR, this means the data is acquired as if there were laser scans, and each agent cannot "see" through walls/obstacles - the LiDAR emulation was extracted from [10] and left unchanged;
- There is communication between agents, which allows exchange of information - merging of their maps. There is an emulated communication range, which is delimited by a square with the agent in its centers and that extends in cardinal directions by the pre-defined range.

In addition, there are three limitations to this environment and will be exposed in the following bullet points:

- The behaviour of the LiDAR emulation should be something to be considered and improved in the future - since it has a minor error due to discretisation of the space;
- the misunderstanding in implementing the DDDQN did not allow to test the environment to its full capabilities - this can be done by improving the observation space and tuning the network for instance;
- the limited implementation of other RL algorithms, such as stable-baselines3[16] and other algorithms available in Ray RLlib[11], which would serve as benchmark - given the incompatibilities and struggles with package versions within the conda virtual environment, this was not implemented given the limited time to develop this project.

4. Results & discussion

The goal was to test the performance of the developed environment, finding the best parameters for learning, even though the hyperparameters of the neural network were not tuned and the values used are from the source code for the DDDQN[3] and can be seen in Table 3. Nonetheless, some other important aspects were tested:

1. influence of stuck verification methods, explained further in this chapter;
2. influence of reward values;
3. influence of number of agents
4. influence of communication

Hyperparameter	Value	Purpose
batch_size	32	batch size
learn_every	10	interval of steps to fit model
update_every	10.000	interval of steps to update target model
alpha	0,0001	learning rate
gamma	0,99	discount factor
epsilon	1,0	exploration factor
epsilon_min	0,01	minimum exploration probability
epsilon_decay	0,99999	exponential decay rate of epsilon
memory_size	100.000	replay memory size

Table 3: DDDQN's hyperparameters

It was verified that for a single agent in a 16x16 dimension map with no obstacles and a maximum numbers of 400 steps, it could get trapped in a cyclic path or in simpler terms, get stuck between positions and for that reason two simple methods to verify if it was stuck were implemented:

- **no stuck verification (no stuck)** - there is no verification and it is shortly designated as ”no stuck” in the following plots.
- **stuck verification method 1 (stuck 1)** - In this method, if the agent has not discovered any new cell for *height*width* of the map steps, then it is stuck. The reason for the *height*width* value is that with 100% certainty any agent can transverse the whole map in that amount of steps, even thought this value is the worst case scenario, since it means the agent passes through each cell one time. For abbreviation, this method will be referred as ”stuck 1”.
- **stuck verification method 2 (stuck 2)** - For this method, an history of the past fifty positions of the agent is saved, if the most frequent one is repeated at least twenty times, then it is considered that the agent is stuck. This method will also be referred as ”stuck 2”.

Three scenarios with different values for rewards were tested, summarised in Table 4. The values for scenario 1 are based on the work [10] and the values for scenario number 2 are based from [19]. Finally, scenario 3 was created from the previous with minor adjustments. It seeks to test the weight of rewarding a bonus as done in [10]and to see the impact of having a different ratio of rewards per new cell explored/penalty for each step taken, thus confirming or not if the agent would choose shorter routes, since each step would become more valuable, having a proportion from the other scenarios.

After training the DDDQN with only one agent in every combination of scenarios and stuck methods in a 16x16 map with no obstacles, the main observations can be summarised as follows:

- all agents learnt to traverse the map in a circular clockwise or counter-clockwise motion which is the most efficient way, even though with no stuck method verification the behaviour persisted and was more frequent than the other two cases;
- the combination of stuck method 2 and scenario 1 proved to have the best performance, providing more coverage in fewer number of steps, as can be seen in Figure 6

The next tests that were done using two and then four agents, in a map with 5 obstacles with a 3x3 shape. Firstly a model was trained with a single agent in the new map and this model was then loaded for the next tests, where only one agent trained the new model. In evaluation moments, all the other agents use the model learnt by that single agent to choose their actions. A single-agent was able to learn how to avoid obstacles, even though this still happened but in a lower frequency.

For the two-agent case, three different values of communication range were tested: 0,0 (no communication), 1,0 and 3,0.

The behaviour that was observed with communication, was that agents would synchronise their actions when in range of each other. Once this happened the agents stopped being able to take independent actions.

For this reason, in the four agent test, no communications were used. In this case, there were agents that got stuck and others that would choose repeatedly to move in the direction of another agent, since there are no collisions amongst agents, this action has no repercussion.

Given the unexpected behaviour of several agents synchronising, the comparison of all the multi-agent cases with no communications regarding the percentage of the map that is explored per step is compared in Figure 7. As expected, the higher the number of agents, higher is the area covered in less steps. In the multi-agent cases, the stuck scenarios are associated with the highest number of steps and that is why the values of the percentage of the area explored decrease and even stagnate with higher number of steps.

Finally, in Figure 8 all the multi-agents approaches are compared and it is evident that the four-agent approach surpassed every other approach. This result was expected, since each agent can cover different parts of the map, resulting in a faster coverage. However, the use of communications were expected to have a better performance.

The first expected result that was verified was the fact that having more agents reduces the number of steps it takes to explore a given space. By opposition, an unexpected behaviour was observed when communications were introduced. A possible explanation can be due to the fact that the agent only receives the explored map as an observation and this has a direct impact in the exploration processing, concretely in the moments where all agents synchronise their decisions when in communication range, since what they see is exactly the same. In order to improve this behaviour, changing the observation space should provide faster learning and a more accurate behaviour, for example using a tuple with the agent’s id, its position, their explored map and information about

Name	Value			Purpose of the reward
	scenario 1	scenario 2	scenario 3	
Movement cost	-0,5	-1	-10	Value that is discounted per step
Collision	-400	-100000	-1000	Negative reward for colliding with an obstacle
Out of bounds	-400	-100000	-1000	Negative reward for getting outside of the bound of the map
Stuck	-400	-10000	-1000	Negative reward when it is stuck between positions
New cell discovered	+1	+10	+10	Positive reward for discovering a new cell
Bonus reward	+400	+10000	+1000	Positive reward when the given percentage of the map is explored (in this case 90%)

Table 4: List of rewards for different scenarios

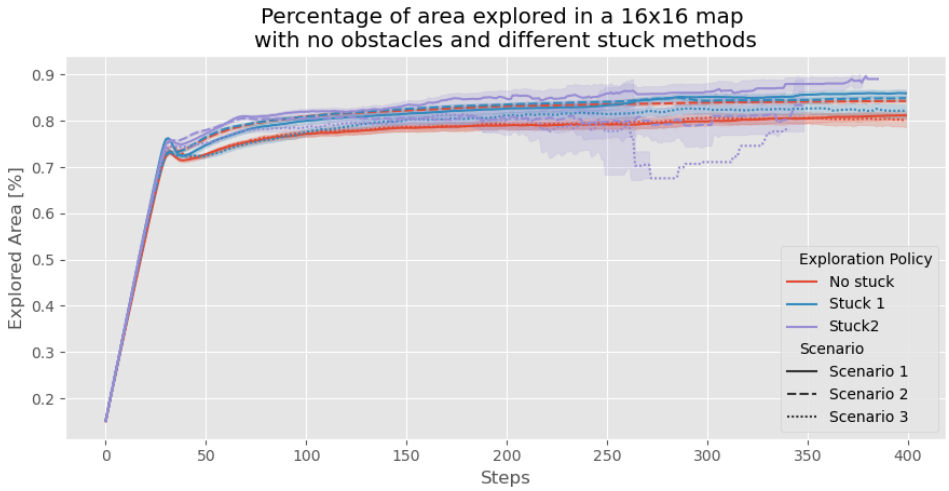


Figure 6: Overview of percentage of the area explored with respect to the number of steps, in a 16x16 area with no obstacles

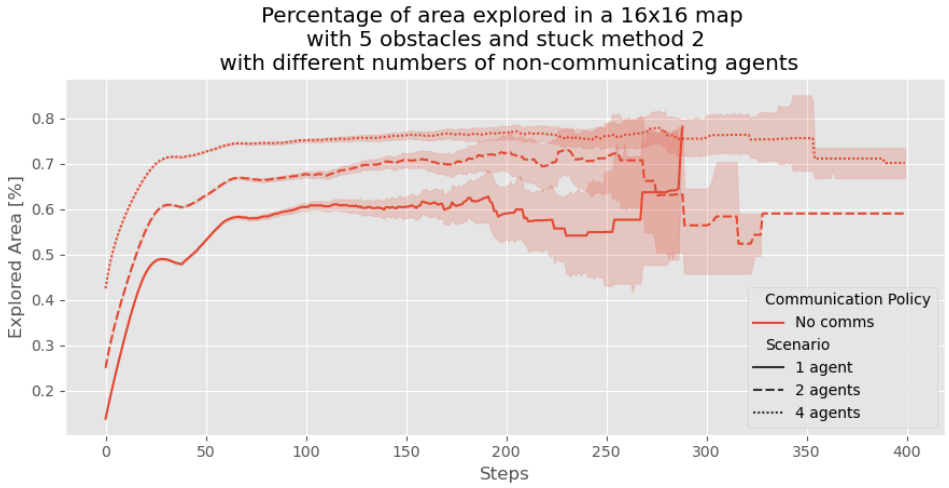


Figure 7: Percentage of the area explored with respect to the number of steps, by various number agents and no communication ranges in a 16x16 area with 5 obstacles

the other agents' positions - a similar approach was used in the ma-gym's Predator-Prey environment. With these changes, each agent should be able to identity themselves and differentiate itself from other agents when they merge their maps, thus avoiding taking the same action as the other agents. Another improvement could be providing a list of frontier points and known obstacles instead of the whole map, in an attempt that the observation space would not be so sparse, hence improving and accelerating the learning process.

Another unexpected behaviour that was seen was the fact that the agents would get stuck between positions. To explain this and as referred in the previous chapter, there is the belief that one crucial step in the implementation of the DDDQN

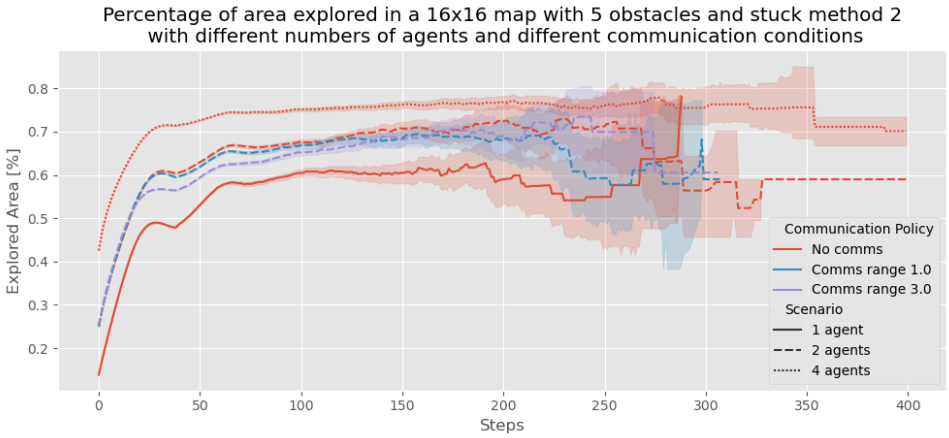


Figure 8: Overview of the percentage of the area explored with respect to the number of steps, in a 16x16 area with 5 obstacles

is missing, namely the stack of frames which provides the notion of motion and past movements. The behaviour where the agents get stuck between positions seems to be deeply related to this missing component. Unfortunately, since this error was detected in at a belated stage of this thesis, it was not possible to present tests and results, but it is a common practice in the implementation of the DDDQN whose inclusion can be noticed by the presented results.

Additionally, the fact that there are no collisions allowed between agents adds another undesired behaviour, it was intended to accelerate the training process but instead it promotes the decision to go against other agents without consequence, meaning that they get stuck choosing the same action of moving in the direction of another agent repeatedly and being counter productive.

Another interesting observation, was the fact that with different ratios of reward for each new cell explored and penalisation for each step taken, the agents seemed to add value to each step taken. For instance, during training and looking at the recordings, when the reward for each new cell explored was 10 and the step cost was -10, the agent learnt in an earlier stage to take shorter routes, whereas when the reward was 10 and the penalty was -1, the agent would take longer and more intricate paths. And the middle term was using the second scenario's reward system, where the reward was +1 and -0.5 was the penalty for each step taken.

Despite these behaviours, it is possible to see that the DDDQN allows a single-agent to learn an optimal path, such as the case where there are no obstacles and the agent learns a circular pattern, in clockwise or counterclockwise motion. With the corrections mentioned in this section and the proper tuning of the neural network, DDDQN shows promising results and the developed environment seems to be simple and robust to develop and test more algorithms.

5. Conclusions

In this work, it was successfully developed a functional multi-agent OpenAI Gym environment for indoor exploration in GNSS-denied environments, in which up to four agents can be simulated with a simple emulated communication system where exploration information can be shared, concretely their maps.

It was possible to verify the impact that the lack of frame stacking in the DDDQN implementation has, precisely, it removes the sense of direction of each agent, thus presenting unexpected behaviour such as having agents stuck between positions. Before getting to this conclusion, simple methods to detect whether agents are stuck were implemented, which overall did not solve the main issue.

Additionally, the impact of several agents was tested and as expected more agents is in general a better solution. Unfortunately the impact of communication could not be properly tested. The results showed that the agents could not identify themselves and when in reach of each other, they synchronised their actions, not being able to search the area properly.

References

[1] N. Boonyathanmig, S. Gongmanee, P. Kayunyeam, P. Wutticho, and S. Prongnuch. Design and implementation of mini-uav for indoor surveillance. pages 305–308. IEEE, 3 2021.

[2] J. Dong, K. Ota, and M. Dong. Uav-based real-time survivor detection system in post-disaster search and rescue operations. *IEEE Journal on Miniaturization for Air and Space Systems*, 2:209–219, 12 2021.

[3] R. Estes. Github repository with implementation of dddqn.

[4] H. A. Foudeh, P. C.-K. Luk, and J. F. Whidborne. An advanced unmanned aerial vehicle (uav) approach via learning-based control for overhead power line monitoring: A comprehensive review. *IEEE Access*, 9:130410–130433, 2021.

[5] GeeksforGeeks. Connected components in an undirected graph.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 1 2018.

[7] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. 10 2017.

[8] M. Juliá, A. Gil, and O. Reinoso. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots*, 33:427–444, 11 2012.

[9] A. Koul. ma-gym: Collection of multi-agent environments based on openai gym. https://github.com/koulanurag/ma-gym, 2019.

[10] D. I. Koutras, A. C. Kapoutsis, A. A. Amanatiadis, and E. B. Kosmatopoulos. Marsexplorer: Exploration of unknown terrains via deep reinforcement learning and procedurally generated environments. *Electronics*, 10:2751, 11 2021.

[11] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica. Rllib: Abstractions for distributed reinforcement learning. 12 2017.

[12] I. Mademlis, V. Mygdalis, N. Nikolaidis, M. Montagnuolo, F. Negro, A. Messina, and I. Pitas. High-level multiple-uav cinematography tools for covering outdoor events. *IEEE Transactions on Broadcasting*, 65:627–635, 9 2019.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2 2016.

[14] S. A. H. Mohsan, M. A. Khan, F. Noor, I. Ullah, and M. H. Alsharif. Towards the unmanned aerial vehicles (uavs): A comprehensive review. *Drones*, 6:147, 6 2022.

[15] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

[16] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations, 2021.

[17] D. Rakesh, N. A. Kumar, M. Sivaguru, K. V. R. Keerthivaasan, B. R. Janaki, and R. Raffik. Role of uavs in innovating agriculture with future applications: A review. pages 1–6. IEEE, 10 2021.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. 7 2017.

[19] A. Seel, F. Kreutzjans, B. Kuster, M. Stonis, and L. Overmeyer. Deep reinforcement learning based uav for indoor navigation and exploration in unknown environments. pages 388–393. IEEE, 4 2022.

[20] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim. Automatic drone navigation in realistic 3d landscapes using deep reinforcement learning. In *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 1072–1077, April 2019.

[21] T. Simonini. Improvements in deep q learning: Dueling double dqn, prioritized experience replay, and fixed. . . . Image source.

[22] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. 9 2015.

[23] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. 11 2015.

[24] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. 8 2016.