Multi-Agent Reinforcement Learning for underwater pipe monitoring

Cyber Physical Systems Project

Alberto Luvisutto

24 February 2022

1 Introduction

The goal of this project, investigated in a real world setting, is to find a strategy to monitor a partially hidden underwater pipe using a swarm of robots, subject to noise on both the movement and the measurements performed.

The approach followed is the implementation of a Multi-Agent Reinforcement Learning model in a model-free, partially observable setting. In the considered decentralized approach, agents have to make independent decisions while interacting with each other and with the environment.

1.1 Setting

The setting is a two-dimensional infinite plane. We chose a 2D setting with the assumption that the agents can maintain a fixed distance from the bottom of the sea, avoiding crashing and, at the same time, allowing the agents to see the pipe with a high value of confidence (probability of recognition 95%).

In our implementation, the underwater pipe is represented by segments of 5 meters length, positioned over the line $y = 0$. The visibility or non-visibility of each segment is randomly sampled at the beginning of each epidose using a Bernoulli random variable with $p = 0.6$. The total length of each episode is also randomly sampled, with the average length of $T = 2000$ time steps (seconds).

1.2 Agents

1.2.1 Characteristics

Each agent has a limited field of view (parameters: $\phi = 1$ rad, $R = 4$ m) in which it is able to sense:

- The presence and orientation of neighbors;
- The presence and orientation of the pipe;
- A visual hint (ex. color of the led positioned on the robot) on how much information the visible agent has on the pipe.

These features are encoded in a discrete finite set of possible states, which will be described in chapter 2.

In order to better suite a Reinforcement Learning setting, also time is discretized and so are the position and velocity update.

1.2.2 Position update

The position of the agent i at time step t is updated according to

$$p_i^{t+1} = p_i^t + v_0 \cdot v_i^t \cdot \Delta t + \omega^t$$

where

- $v_0 = 0.3$ m/s is the constant speed, equal for all agents
- $\Delta t = 1$
- $||v_i^t|| = 1$
- ω^t is a 2D vector of Gaussian noise sampled from $\mathcal{N}(0, 0.01)$.

1.2.3 Velocity update

The velocity of the agent i at time step t is updated according to

$$v_i^{t+1} = R(a_i^t + \theta^t) \cdot v_i^t$$

where

- R is a rotation matrix;
- a_i^t is a rotation angle and corresponds to the action selected by the agent;
- θ^t is a scalar Gaussian noise sampled from $\mathcal{N}(0, 0.08)$.

1.3 Goal of the project

The goal of the project is to find an optimal policy (one for each agent, since the problem is decentralized) in order to follow and monitor the hidden pipe. Since we are following a RL approach, we try to achieve the task maximizing the sum of the expected rewards received by the agents. The objective is to cover the whole length of the pipe and monitor all its visible sections.

In order to do this, a swarm approach is used to try to:

- Favour implicit spreading of information, through optical cues on the level of information of preceding agents;
- Improve policies through imitating behavior between agents;
- Explore the state space in a more convenient way, thanks to collective motion and imitating behavior.

2 Model

The approach used is Temporal Difference with Expected SARSA, a classical Reinforcement Learning algorithm (no use of Artificial Neural Network to perform function approximation) that tries to approximate the so called Quality function Q with a matrix following this update, performed at every time step t:

$$Q_i(s_i^t, a_i^t) \leftarrow Q_i(s_i^t, a_i^t) + \alpha\{R_i^{t+1} + \mathbb{E}_\pi[Q_i(s_i^{t+1}, a_i^{t+1})|s_i^{t+1}] - Q_i(s_i^t, a_i^t)\}$$

where

- s_i^t is the state of agent i at time t;
- a_i^t is the action of agent i at time t;
- R_i^{t+1} is the reward received by agent i at time $t+1$;
- π is the target policy;
- α is the learning rate.

The process is repeated for 16000 episodes; the length of each episode is variable, due to a survival probability $\gamma = 0.9995$, so that with probability $1 - \gamma$ the system gets to a terminal state. At the end of each episode, agents are re-positioned at the beginning of the pipe, with random initial position and velocity (with certain restrictions), while keeping the current approximation of Q.

In order to properly measure performances of the system, the total visible section of pipe is calculated based on the maximum possible distance coverable by the agent if it perfectly followed the pipe for the whole length of the episode.

2.1 States

The states are three dimensional tuples (sn, sp, si), and each one of them encode the information as follows (for each agent i in each time step t).

The neighbors state sn is calculated as:

$$sn_i^t = \begin{cases} \arccos\left(\frac{P_i \cdot v_i^t}{||P_i||}\right) & \text{for} \quad P_i \cdot (v_i^t)_\perp \geq 0 \\ -\arccos\left(\frac{P_i \cdot v_i^t}{||P_i||}\right) & \text{for} \quad P_i \cdot (v_i^t)_\perp < 0 \end{cases}$$

with P_i as the weighted normalized average velocity of the visible neighbors. The weight of the neighbor depends on how much information it has on the pipe.

This state represents the angular difference between the agent's velocity v_i and its neighbors. A state like this is usually present in tasks that try to achieve some sort of collective motion, in order to give the possibility to non-informed agents to learn an imitating behavior (usually alignment with informed ones).

After the computation, the value of the state is discretized on a set of 33 states. 32 of these corresponding to the equally spaced discretization of the interval $[-\pi, \pi[$. An extra state is added to take into account the case in which there are no agents in the field of view.

The pipe state sp is calculated similarly to sn, but in this case we compute the angular difference between the agent's velocity v_i and the measured orientation of the pipe, represented by the normalized vector O.

$$sp_i^t = \begin{cases} \arccos\left(\frac{O^t \cdot v_i^t}{||O^t||}\right) & \text{for} \quad O^t \cdot (v_i^t)_\perp \geq 0 \\ -\arccos\left(\frac{O^t \cdot v_i^t}{||O^t||}\right) & \text{for} \quad O^t \cdot (v_i^t)_\perp < 0 \end{cases}$$

Note that the information on the orientation of the pipe is kept in memory by the agent in order to be used also in time steps in which the pipe is not visible. This value is updated in each time step in which the pipe is visible by the agent, and the update is performed using a running mean with a forgetting factor $\beta = 0.99$.

The measurement performed by the agent is subject to Gaussian noise sampled from $\mathcal{N}(0, \pi/16)$. This means that, without any running average on the measurement, the agent would misclassify the pipe state one out of three time steps, on average.

The information state si is more similar to an encoding state; in fact, it is computed as

$$si_i^t = \begin{cases} 0 & \text{if agent sees the pipe and is close to it} \\ 1 & \text{if agent sees the pipe on the right edge of its fov} \\ 2 & \text{if agent sees the pipe on the left edge of its fov} \\ 3 & \text{if agent has just lost the pipe} \\ 4 & \text{if agent has lost the pipe since a long time} \end{cases}$$

The previously reported states are directly connected to the weight system with which the level of information of an agent is assessed, during the calculation of P_i for state sn. In fact, the weight of an agent is determined as:

$$w_i = \begin{cases} 0.8 & \text{if } si_i = 0 \\ 0.4 & \text{if } si_i = 1 \quad | \quad si_i = 2 \\ 0.2 & \text{if agent sees an informed agent} \\ 0 & \text{otherwise} \end{cases}$$

with "informed agent" meaning an agent with $w_i > 0$.

2.2 Actions

At each time step t, based on the previously defined states, each agent i selects and performs an action a_i^t, choosing from a predefined set of 7 elements, consisting in turning angles, corresponding to:

$$A := \left\{ -\frac{3\pi}{16}, -\frac{2\pi}{16}, -\frac{\pi}{16}, 0, \frac{\pi}{16}, \frac{2\pi}{16}, \frac{3\pi}{16} \right\}$$

The action is selected following an ϵ-greedy policy:

$$a_i^t = \begin{cases} \text{argmax}_{a'} Q_i(s_i^t, a') & \text{w.p.} \quad 1 - \epsilon \\ \text{random action} & \text{w.p.} \quad \epsilon \end{cases}$$

where ϵ is the exploration rate.

2.3 Rewards

The received reward R_i^{t+1} is exclusively based on the position and orientation of the agent with respect to the pipe. It is computed as

$$R_i^t = \begin{cases} \cos(\beta_i - \zeta) - 1 & \text{if agent sees the pipe} \\ -1 & \text{otherwise} \end{cases}$$

where:

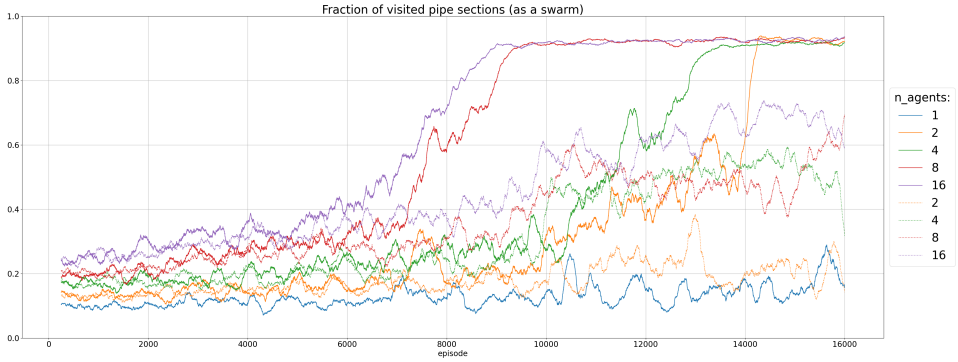
- β_i is the angle defining the orientation of agent i;
- ζ is the angle defining the real orientation of the pipe.

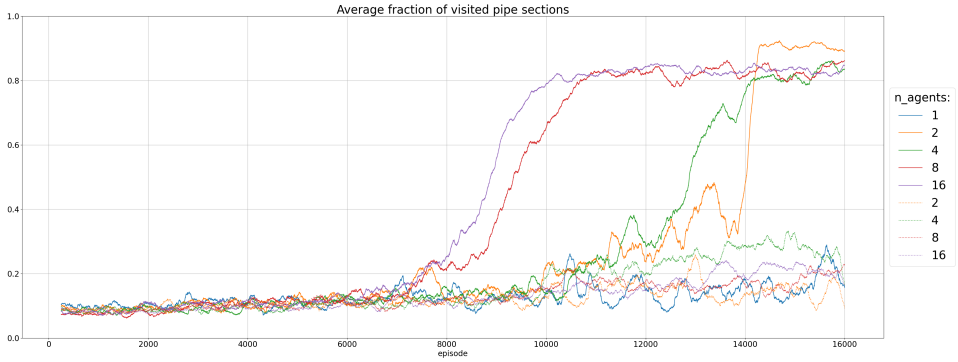
Note that there is no reward to encourage a collective behavior.

3 Results

In order to show that there is an advantage on actively using a swarm, and to show that there is some sort of collective motion due to the influence that agents have towards one another, "normal" simulations have been compared to "baseline" runs in which we take away the neighbor state sn, leaving the agents "blind" with respect to their neighbors.

The simulations performed show that, in "normal" simulations, agents independently learn a policy that achieves good results, both as a swarm (first graph) and independently (second graph).





It's also visible from the previous plots, that:

- Agents working together (plain line) learn a good policy, differently from the baseline case;
- Increasing the number of agents involved decreases the number of episodes required to converge to an optimal solution.

4 Validation

Since the code has been developed in Python, the process of validation has been conducted using the Moonlight library extension for Python.

Through validation on single runs, I wanted to observe some properties that were difficult to investigate without the use of STL.

The investigated properties are:

- Distance between agents: we want to show that agents learn to stay together, even though we don't encourage this behavior. This is difficult to observe just looking at plotted trajectories, since episodes are very long. What we want is to observe a distance that stays lower than the radius of the field of view (4 meters).

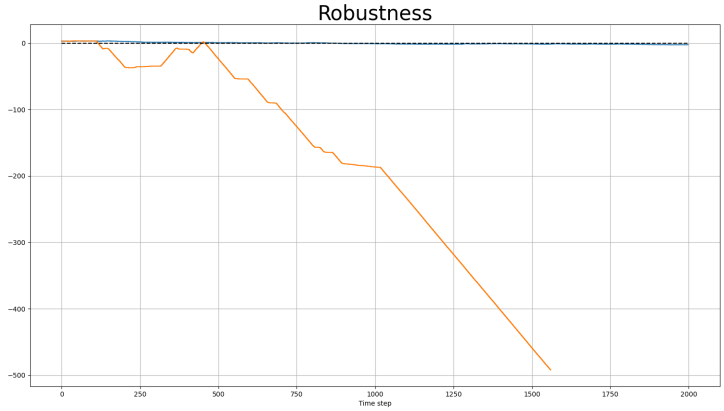
  In STL:

  $$\psi_1 = G_{[0,T]}(d(p_1, p_2) \leq 4.0)$$

  I hereby report some values of the robustness for the case of 2 agents, in a simulation performed with $\gamma = 0.999$. It's clearly visible when the algorithm starts to converge to a solution, which also corresponds to improved collective motion.

Episode	Robustness
1	-40.178
2500	-22.458
5000	-190.896
7500	-16.063
10000	-42.057
12500	-39.702
15000	-90.642
17500	3.059
20000	1.809
22500	-1.842
25000	3.117
27500	2.665
30000	1.254
31900	1.312
32000	1.629

  Just as an example, I here report a plot comparing the robustness value for an episode at the end of the learning (green line) and one at the beginning (orange line). (This graph is obtained plotting G[0,1] instead of G[0,T] to show the robustness value on the singular time steps).



- Consequences of seeing an intelligent agent: we wanted to show that the agent learns to follow an intelligent agent when it sees one. Again, this is very difficult to see just looking at single trajectories, so we tried to use an STL formula, here reported.

  We tried with two different formula.

  The first one is:

  $$\psi_2 = G_{[0,T]}((w_i == 0.2) \rightarrow F[0,20](agent\_see))$$

  And the second one is:

  $$\psi_3 = G_{[0,T]}((w_i == 0.2) \quad U[0,20](agent\_see))$$

  in which we are trying to prove the fact that not only the agent *eventually* gets to see the pipe again, after seeing an intelligent agent, but also never leaves the "intelligent state" before seeing it.

  With the first formula the results are not very useful for what we tried to achieve. In fact, the given predicate is true whenever $w_i \neq 0.2$, that is, also when the agent is not seeing the pipe and not seeing any smart agent ($w_i == 0$). This means that, for the initial episodes in which the agents doesn't see the pipe most of the time, the robustness often has a misleading (meaning we don't interpret it in the right way) high value.

  Moreover, when we look at the value of the robustness on the final episodes of the learning, both the noise on the movement of the agent and the small field of view play a big role. In fact, it becomes very difficult for the agent to keep the preceding smart one in its field of view at *all* time steps until it sees the pipe again. Nevertheless, the agent still manages to follow in some way the preceding agent, as also proven by the values of robustness previously observed about distance.

The issues reflects also on the second formula; nevertheless, if we take a look at the robustness values, when computing them for each time steps we can observe different behaviors. While at the beginning of the learning the robustness gets as low as it can get, due to the non-satisfaction of the second part of the formula, at the end of the learning, if we get a negative robustness value, most of the times it is due to the non-satisfaction of the first part of the formula, which as previously stated is inevitable in most cases.