Master 1 - Semester Project

Intention recognition under uncertainties for human-robot interaction

Nicolas Talabot
nicolas.talabot@epfl.ch

Supervised by
Mahdi Khoramshahi
Laura Cohen
Prof. Aude Billard

June 9, 2017

Contents

Chapter 1

Introduction

Collaboration consists of multiple individuals interacting and working together in order to achieve a shared goal. As they have to joint their attentions and actions in order to efficiently fulfil the task at hand, a need of coordination between them arises. Anticipating their partners' actions and intentions, as well as planning their own, allow them to answer this, and to perform complementary actions toward their goal. Research suggests that humans rely on different mechanisms to achieve this coordination in joint actions [1].

As robots and autonomous systems are becoming increasingly present in workplaces, humans and robots need to achieve this kind of coordination to efficiently, and safely, work together. In these scenarios, the robot is often fulfilling the role of an assistant to the human user, who acts as a "leader" aware of the task to be accomplished. For example, we can think of a heavy load, requiring the collaboration of a human and a robot, to be moved to another room, but only the human knows to which one. The robot has to understand the correct task in order to provide a good assistive behaviour. There is then a need to infer correctly the human's intention.

However, what could seems obvious to humans might be hard to interpret for robots: inference becomes a hard problem because of the uncertainties. They can be of different forms and sources, for instance it could be actions from the human irrelevant to the task that could be misinterpreted by the robot, or some errors in the robot's readings that might lead to false conclusions or ambiguity. Two mechanisms of robot behaviour have been studied for human-robot collaboration: the use of gaze to disambiguate, and the development of an autonomous system to rectify communication breakdowns [2]. Moreover, a recent work proposes an anticipatory control based on visual clue (the direction of the human's gaze) and speech recognition for efficient collaborations [3]. In this work, we are interested to investigate whether the ability to recognize the intention of other agents can be learned during a specific interaction, despite the uncertainties. To do so, we propose to study the behaviour of two learning agents over a simple collaboration problem, using Reinforcement Learning.

Chapter 2

Background and related works

In this section, we provide a quick overview for the background and related works. To propose our preliminarily learning method for intention-recognition, we focus on simple settings for the collaborative task. As task with ambiguity in its final goal, we choose the "Double-Slit" problem, to be solved using Reinforcement Learning. In the following sections, we present the theoretical foundations behind the project.

2.1 Reinforcement learning

The following overview for Reinforcement Learning is based on the work of Sutton and Barto [4]. For a deeper and more comprehensive description of Reinforcement Learning, please refer to their textbook. Reinforcement Learning (RL) is an area of machine learning where an agent (capable of performing a set of actions), or an algorithm, is learning a behaviour. The agent exists in an *environment* where it can perform an action, change its state, and collect rewards. The agent learns to achieve a certain goal by maximising its reward through its interaction with the environment; more epsecially, by trial-and-error.

It revolves around the notions of *states*, *actions*, and *rewards*. The state comprises all the informations available to the agent about its current situation, and the current status of the environment. For instance, for a simple walking robot it could be its current GPS position. The action represents the possible controls, or actuations that the agent can perform. We need to note that the set of possible actions might be dependent on the current state. For our walking robot example, possible actions could then be "turn around", "make a step", or even "do nothing". Finally, the reward is simply a numerical number. It is a signal given by the environment that the agent is able to interpret as a gain, or a cost. Continuing with the example, a positive reward could be given when it has reached a certain targeted position, and a negative one might be given for every step made as it requires energy. The aim of the agent is to maximise its cumulative reward in the long run. Thus, the reward function, which assigns rewards to the agent, defines the goal of the agent. Our walking robot will then try to reach the targeted position in the least steps possible.

The whole learning process is fractioned in time steps. At the beginning, the agent can assess the current state s. Then, it has to perform an action a (as previously said, "doing nothing" might also be considered as an action) on the environment. This causes a transition from the current state s to the next one s' (it is possible that $s' = s$), that can be deterministic, as well as stochastic.

Additionally, this transition is accompanied by a reward r given by the environment to the agent:

$$s \to a \xrightarrow{r} s' \to a' \xrightarrow{r'} s'' \to ...$$

This cycle continues until a terminal state is reached, or might persist and never stops. A trial from the starting state to a terminal one is called an episode. If the episode never finishes, it is said to be a continuing task. However, for this project, we will consider episodes that terminates.

During this interaction, the agent uses the information gained by the rewards it obtains to create a mapping from states to actions in order to maximise its reward. This mapping is called a policy π. There exists at least one which is optimal, designed by π^*, for which the expected accumulated reward is maximal. This is called the return:

$$G_t \doteq \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \,,$$

where t is the current, and T is the final time step (for continuing tasks, $T \to \infty$). Moreover, we have introduced a parameter $\gamma \in [0, 1]$, called discounting. It controls how far sighted the robot is by changing the weighting of the future rewards (the value of 1 cannot be used in case of a continuing task).

A sum up of the process is illustrated in Fig. 2.1. In addition, we can extend this model to be used with two learning agents, for intention inference. To achieve this, we add a second agent that will also perform actions on the environment, and receive rewards. Moreover, one agent will have an objective while the other will try to guess it, and the agents will share a common state. The resulting process is illustrated in Fig. 2.2.

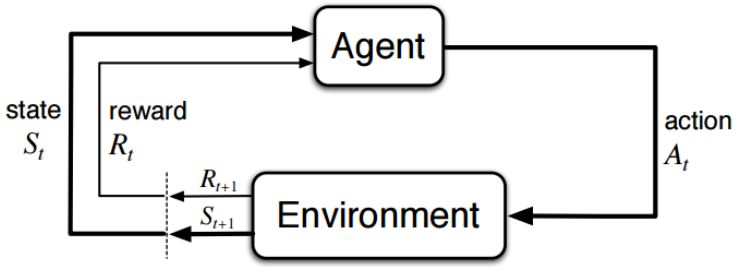


Figure 2.1: Illustration of how Reinforcement Learning works [4].

Finally, we introduce two last important notions of RL. The first one is called the state value function $v_\pi(s)$, and represents how good for the agent it is to be in the given state s. It is defined as the expected return from the state s, while following the policy π:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \,|\, S_t = s]$$

The second notion is analogous, it is the action value function $q_\pi(s, a)$, and represents how good for the agent it is to take action a in state s. Similarly, it is defined as the expected return from the state s, after taking the action a and then following the policy π:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \,|\, S_t = s, A_t = a]$$

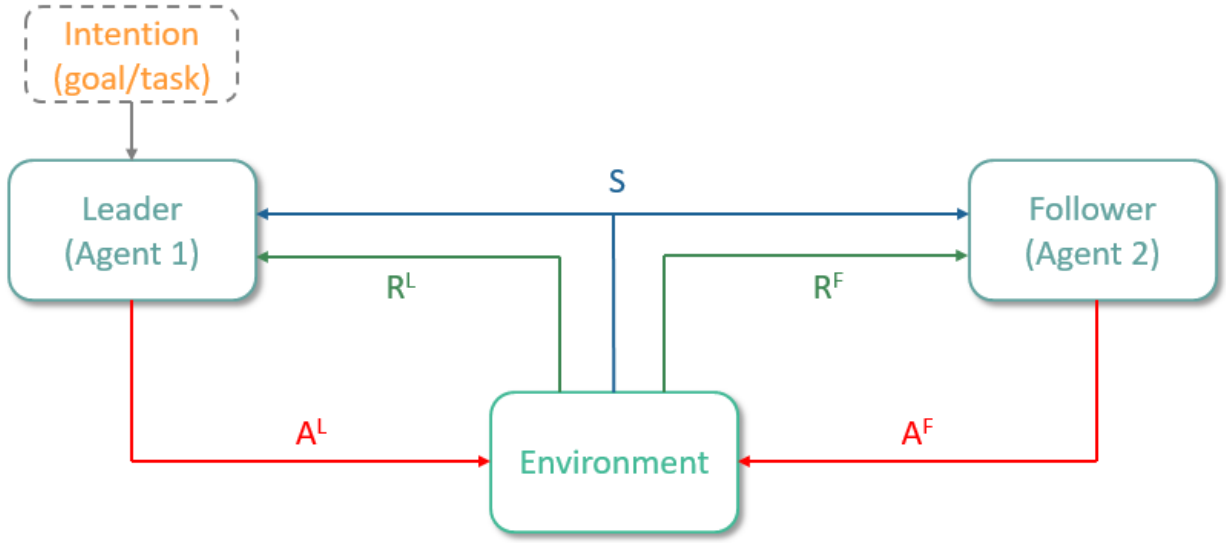


Figure 2.2: Illustration of how we extend the initial RL framework to accept two agents: a leader that has an intention, and a follower. The two agents share a common state.

The agent has then 3 different ways of estimating the optimal policy. First, it can try to estimate the value functions in order to implicitly derive the policy from them (by taking actions that make the agent transit to higher valued state). To estimate the optimal policy, this method alternates between estimating the value function, and improving the derived policy. This is called Generalized Policy Iteration (GPI). Another possibility is to directly estimate the optimal policy without computing the value functions. The last method consists in combining both by estimating both the value functions and the policy explicitly. Multiple RL algorithms has been developed based on these 3 different methods. In this project, we will use two different techniques: SARSA, which tries to estimate the action value function, and Policy Gradient, which directly estimates the parametrized policy.

2.1.1 SARSA

SARSA is a Temporal-Difference (TD) Learning algorithm, and was used for the majority of the project. TD algorithms are variations of GPI, in the sense that they look for the optimal policy by estimating the value functions. Moreover, TD techniques make approximations based on other approximations, that is they *bootstrap*. SARSA approximates the action value function $q_\pi(s, a)$, and keeps its current estimates $Q(s, a)$ in a table. These values are only updated after that the agent performs the action a in state s, and receives a reward. The update rule is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma Q(s', a') - Q(s, a) \right]$$

We can notice the discount factor γ, as well as a new parameter $\alpha \in [0, 1]$. It is the learning rate, that is, how fast the estimate $Q(s, a)$ is moved toward the new estimation. For a value of 0, the estimation never changes and the agent cannot learn. If the value is too high, the estimation might never be able to stabilize and the solution becomes unstable. Thus, it is important to choose an adequate number. In addition, we observe that the term $Q(s', a')$ depends on both the next state s', and the action a' that the agent will perform in this new state. The name SARSA actually

comes from this (State-Action-Reward-State-Action).

To estimate the action values, the agent needs to interact with its environment. Then, we perform multiple episodes in which the agent will update its estimates. Furthermore, it is necessary to explore in order to visit every state-action pairs (s, a). Thus, an explorative policy is used during learning: the ϵ-greedy policy. It answers the exploration/exploitation trade-off by selecting the greedy action (action considered as the best available) with a probability of 1-ϵ, and a random action amongst the available ones with a probability of ϵ.

Finally, to improve the speed of the learning, we add *eligibility traces* to SARSA. These traces can be viewed as back-propagating rewards, in order to assigned some credit or blame to previous actions. The value of the back-propagation is moderated by a factor $\lambda \in [0, 1]$. The full algorithm is given in Alg. 1.

Algorithm 1 Tabular SARSA with eligibility traces [4]

Initialize $Q(s, a)$ arbitrarily, and $e(s, a) = 0$, for all s,a
Repeat (for each episode):
    Initialise S
    $e(s, a) = 0$, for all s,a
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each time step in the episode):
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
        $\delta \leftarrow r + \gamma Q(S', A') - Q(S, A)$
        $e(S, A) \leftarrow e(S, A) + 1$
        For all s,a:
            $Q(S, A) \leftarrow Q(S, A) + \alpha\delta e(s, a)$
            $e(S, A) \leftarrow \gamma\lambda e(S, A)$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

2.1.2 Policy Gradient

During the last part of the project, we have tried to implement the Policy Gradient. It is a method that learns a *parametrized* policy, that can choose actions without the need to consult a value function. We denote the parameters of the policy by $\theta \in \mathbb{R}^n$. The probability of taking action a in state s thus becomes $\pi(a|s, \theta) \doteq p(A_t = a|S_t = s, \theta_t = \theta)$. As we use this technique with continuous state and action, we decide to us a linear Gaussian policy, defined as:

$$\pi(a|s, \theta) \doteq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \theta))^2}{2\sigma^2}\right),$$

where σ is the standard deviation of the Gaussian, and $\mu(s, \theta)$ its mean, defined as:

$$\mu(s, \theta) \doteq \theta^T \phi(s),$$

where $\phi(s)$ is the state feature vector. This vector is constructed by putting multiple Radial Basis Functions (RBFs) in the state space, and then computing each memberships of the current state.

The feature membership depends on the mean c_i, and the covariance matrix Σ_i of the i^{th} RBF:

$$\phi_i(s) \doteq \exp\left(-\frac{1}{2}(s-c_i)^T\Sigma_i^{-1}(s-c_i)\right)$$

Finally, to compute the gradient ascent over the parametrized policy, we use the REINFORCE algorithm with the following update rule:

$$\begin{aligned}\theta_{t+1} &\doteq \theta_t + \alpha\gamma^t G_t \nabla_\theta \, log\, \pi(A_t|S_t,\theta) \\ &= \theta_t + \alpha\gamma^t G_t \frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\end{aligned}$$

where

$$\frac{\nabla_\theta \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} = \frac{a-\mu(s,\theta)}{\sigma^2}\phi(s)$$

For the parameters, we notice that we still have the discount factor γ and the learning rate α, but we don't use eligibility traces anymore. In addition, we also have σ, the standard deviation of the Gaussian policy. The overall algorithm is given in Alg. 2.

Algorithm 2 Episodic REINFORCE [4]

Input: the differentiable Gaussian policy parametrization $\pi(a|s,\theta)$
Initialize policy weights θ arbitrarily
Repeat (for each episode):
    Generate an episode S_0, A_0, R_1,..., S_{T-1}, A_{T-1}, R_T, following $\pi(\cdot|\cdot,\theta)$
    For each time step of the episode $t = 0$,...,$T-1$:
        $G_t \leftarrow$ return from time step t
        $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \, log\, \pi(A_t|S_t,\theta)$

2.2 Double-slit problem

The experimental setup that we use is the Double-slit problem, that has been used in the study of stochastic optimal control [5]. The problem is the following. There is a particle living in a 2D world, which is moving with a constant speed in the horizontal direction (that we label x). At some point, there is a vertical wall with two slits, and the particle has to pass through one of the slit. To do so, the agents have control over its vertical speed (we label this direction as y). There is however noise in this direction, so that the particle is constantly deviated from its path. We illustrate this in Fig. 2.3.

This setup is interesting because it is simple yet suitable for our problem. It simulates two tasks (the slits to reach), with some underlying ambiguity as they are relatively close to each other. Moreover, the uncertainties are represented by the noise deflecting the particle in y. Therefore, we might use this experiment as our problem of intention inference: one agent will know the correct slit, and the second one will have to infer it under the presence of noise. Through this report we may refer to the slits as doors, goals, or even tasks.

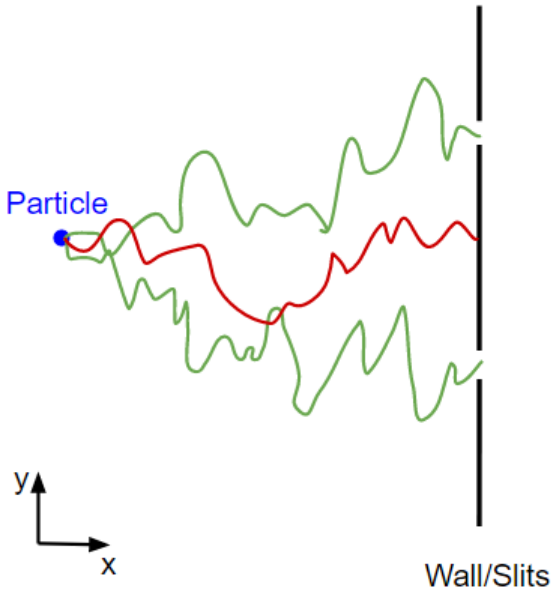


Figure 2.3: Illustration of the Double-slit problem. We show three examples of possible trajectories from the current particle position, two of them (in green) lead correctly to a slit, one (in red) leads incorrectly to the wall.

Chapter 3

Experimental setup

This chapter present the implementation of the double slit problem as a Markov decision process for Reinforcement Learning. We explain how we make the agents learn, and introduce the different kind of scenarios that we study.

3.1 Markov model

3.1.1 State-space

The state that we use for RL is simply the (x, y) position of the particle in the 2D world. However, because SARSA stores the estimates of the action value function in a table, we cannot have an infinite world. Thus, we limit the particle to a 100-by-100 world: $x \in [0, 100]$ and $y \in [-50, 50]$. The particle always starts the episode at the origin $(0, 0)$, whereas the wall and slits are situated at $x = 100$. The slits' centres are at $(100, 25)$ and $(100, -25)$, and they both have a width of 6. Moreover, the particle cannot leave the world, so that if the agents try to push it off, the particle will simply stay against the boundary.

As RL works in time step, the time need to be fractioned. In addition, SARSA requires discrete spaces of state and action. Therefore, we have to discretize the world into a gridworld. We tried different values of sizes for the grid's cells, and in order to have a nice learning and tractable results, we fixed the cells to 2-by-2. Finally, in order to implement the constant speed in x, we make the particle move one cell to the right at each time step.

3.1.2 Action-space

The agents can interact with the environment by moving the particle. Nevertheless, as we have fractioned the time, they don't control the particle vertical speed anymore, but rather directly displace it: $\frac{dy}{dt} \cdot \Delta T \sim \Delta y$. Then, the actions available to the agents are movement in y. We bound them to $a \in [-4, 4]$ in order to represents more realistic behaviour, and to avoid large jumps over the grid.

We eventually add the noise to these actions to get the final displacement in y. For the implementation of SARSA, the noise is uniformly chosen among $\{-2, 0, +2\}$. As the spaces are continuous for the Policy Gradient, we chose to implement the noise as a white Gaussian noise with a standard deviation of 2 (the cell size).

3.1.3 Reward function

As the reward function implements the goal of the agent, it has to answer to our expectation. First of all, to make the agent learn to pass the particle in the slits, we give a penalty of -100 for hitting a wall, and a reward of +100 for going through the correct door. The "correct" door is defined as being the task that the leader has to realise, and that the follower has to infer. Moreover, in order to force the agent to minimise its actions on the particle, we add a movement cost. Thus, the agent receives a penalty equals to $-|a|$, that can represent the "energy" needed to perform the action. In addition to that, we chose to set the discount factor γ to 1, that is, no discounting. We have taken this choice as it makes more sense to sum every reward to assess the performances of the agent (goal achieved vs. energy consumption). Thus, the agent should also use the total accumulated reward in order to learn. Finally, to force the follower agent to help the leader, we have changed its reward function so that it actually receives the leader's movement cost instead of its own. In this way, it would have to anticipate the leader's actions in order to perform them in advance, to maximise its reward.

3.2 Learning method

The agent learns by trial-and-error directly by interacting with the environment, the double slit problem. We repeat many episodes until the results are found to be relatively constant. To evaluate the learnings, we plot different informations. We start by the learning curve, which is the accumulated reward that the agent obtains in each runs of the learning. This is useful as we can directly see if the agent tended to increase its reward, thus if it learnt. Then, after the learning is finished, we plot the state value functions (in the case of SARSA) over the grid, as well as the actions that the agent takes in each states (that is, the policy). As performances metric, we simply use the accumulated reward of the agent during an episode. Then, we make the agent perform again on the double slit problem after making its policy greedy by cancelling all exploration.

As we have seen, the algorithms that we use have hyperparameters that have to be manually tuned. To do so rigorously, we should perform a gridsearch with cross validation over all the possible combinations. However, as the learning is not fast, this would represent an important investment in time. Additionally, we are not necessarily looking for the optimal solution, with the best of the agents. We rather wants to see how they can learn to recognise an intention. Thus, we accept any working solutions, as long as the results of the learning are correct. Moreover, as it will be discussed next, we have different kind of scenarios that might cause the optimal set of parameters to change. But, it is possible that we should keep the same set of parameters over the scenarios to effectuate a fairer comparison. Thus, we prefer to find a solution that fits all. Finally, to find the parameters of SARSA, we have performed multiple tests until a set was manually found to be satisfactory. The resulting values are $\alpha = 0.1$, $\epsilon = 0.1$, and $\lambda = 0.5$. For the Policy Gradient, as it will be shown in Results, the learning was not working well. Thus, many different sets of parameters gave the same results. The one we chose here is then only an example: $\alpha = 0.1$, and $\sigma = 1$ (as a reminder, σ is the standard deviation of the linear Gaussian policy).

3.3 Scenarios

In order to perform different tests to compare the results, we slightly modify the original Double slit problem. We sum up the different versions here, as multiple "scenarios".

3.3.1 Single-agent

Initially, there is only one agent acting on the particle. The first scenario is the basic Double slit problem that we have presented: the agent moves the particle in the y direction in order to pass it through one of the 2 slits. This first version allow us to verify that our implementation work, and to test different parameters in order to chose their values. As the agent can make use of both slits, we designate this as a "multi-goal" scenario. After this, in order to simulate the separate tasks, we change to a single-goal scenario. Now, there is only one correct slit at each episode, and the other is considered as a wall. Therefore, the agent now learns two different policies, one to realise each task.

3.3.2 Multi-agent

Eventually, we introduce a second agent to the problem. Both agents now act on the particle, and it is their resulting joint action that moves it. However, this time we only consider the single-goal case, that is, a specific slit is chosen at each episode. The first agent is the leader, and has two policies (one for each slit, as the single agent). He is then aware of the correct slit. The second agent is the follower, and it has only one policy that has to work for both tasks. Thus, it would need to infer the leader's goal to act correctly on the particle.

Nevertheless, we divided this into 2 scenarios, depending on the leader. The first one consist of a "fixed" leader that has already learned how to go through the slits before the addition of the follower. Moreover, it does not continue to learn with the follower. It is implemented by utilizing the results of the single-agent single-goal scenario. The second multi-agent scenario is using an "adaptive" leader. This agent does learn in parallel of the follower, and is not initially implemented from the results of the single-agent.

Chapter 4

Results

4.1 Single-agent

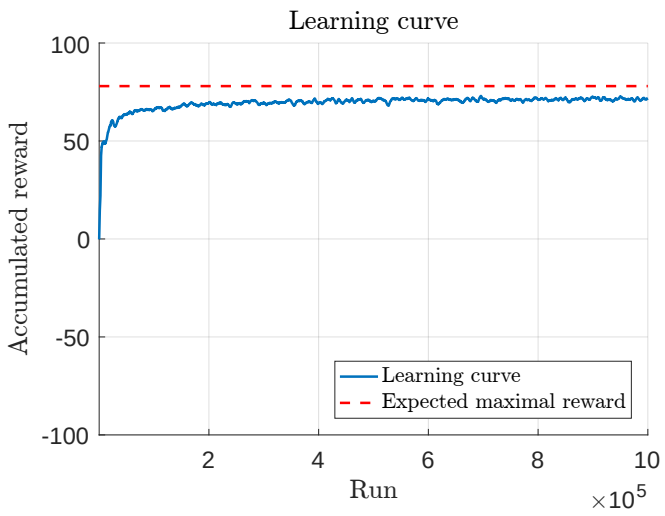
In this section, we present the results of the learning when only a single agent acts on the particle. We further divide the results in two scenarios: multi and single goal.

4.1.1 Multi-goal

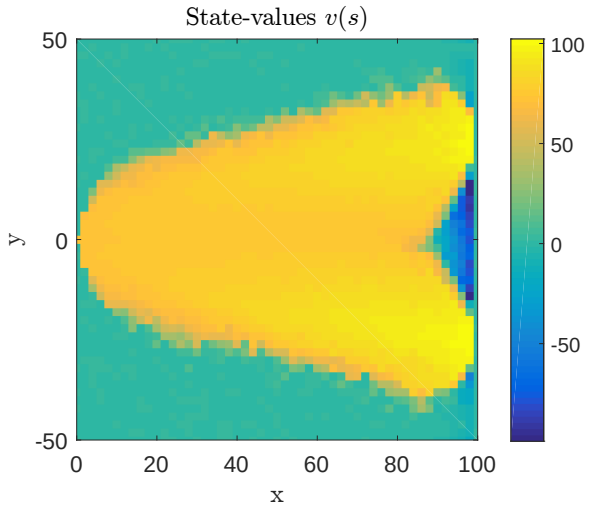
The first scenario is the basic double-slit problem: there are several terminal states (i.e., final targets) that fulfil the task. In our case, the agent has to pass the particle through either slit to receive the reward at the end of the grid. The learning curve can be found in Fig. 4.1a. It is the accumulated reward obtained during the learning, versus the run. The expected maximal reward represents the optimal average reward that the agent is able to get on this problem. That is, it's the final reward of the slit minus the cost of movements needed to reach that slit. The curve does not reach this value (of approximatively 78) because of the exploration during the learning. However, once the policy is made greedy, the agent is able to attain it (see Fig. 4.1d, the second value is the standard deviation). The state-value function $v(s)$ that the agent has learned is plotted in Fig. 4.1b. We notice that high values seem to "spread" out of the 2 slits in conic shapes: while inside this region, the agent will be able to pass the particle through a slit, thus receiving a high reward. We need to note that the external regions in green (top and bottom left, for instance) that have values around 0 are simply unexplored during the learning. However, the particle won't enter them, so we can simply disregard them in most plots. The actions that the agent has learned are plotted in Fig. 4.1c. Here we can see that the agent tries to keep the particle inside a wide central region at the beginning, and starts to push it towards the closest door when getting closer to the walls. Finally, to observe the results of the learning, we run 200 times the problem with the newly learned policy, and plot the resulting trajectories in Fig. 4.1d. The two colours are simply to differentiate the trajectories that end in the top slit from the ones that end in the bottom slit. By looking at them, we are able to see this initial spread due to noise, followed by a converging toward the closest door.

4.1.2 Single-goal

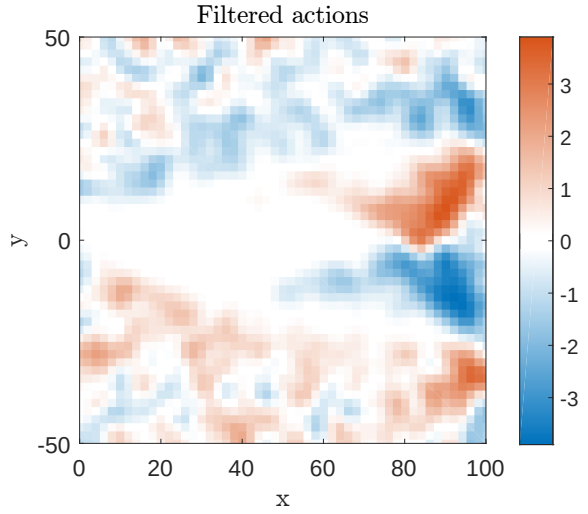
We are now interested to see how the agent performs for a particular goal rather than a mix of both. Hence, there is now only one terminal state that fulfil the task. In our case, going only through the designated door will result in the positive reward. Now the agent has to learn two different policies: one for the top slit, and another for the bottom slit. Thus, there are going to be



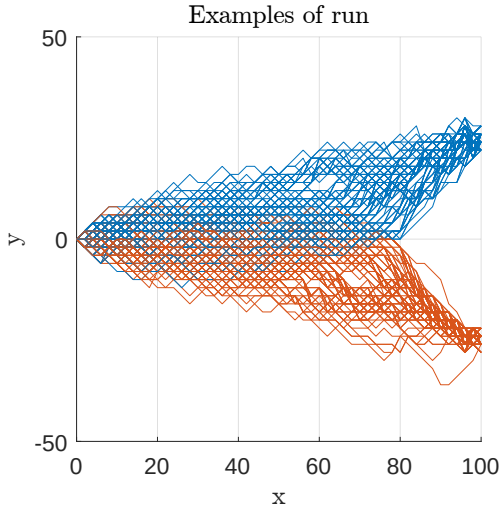
(a) Averaged learning curve over a moving window of size 5000. The curve is unable to reach the maximal reward because of the exploration



(b) State-values over the grid. The external regions in green that are worth 0 are unexplored.



(c) Actions over the grid, filtered with a 5-by-5 Gaussian filter. The agent keeps the particle in a central region before moving it toward the nearest slit.



(d) Trajectories of 200 runs after the learning. The colour of the trajectories are based on slit that the particle goes through (Average reward, mean ± std: 79.20 ± 5.70)

Figure 4.1: Results of the learning for the single agent in a multi-goal scenario

2 learning curves, as well as two state-value functions and actions. We first start with the learning curves, both plotted in the same graph in Fig. 4.2. The two curves are superimposed: the learning is relatively equivalent in both cases as the problem is symmetric. This allows us to conclude that there is no bias toward any goal. Moreover, if we recall the learning of the previous scenario (Fig. 4.1a), we observe a drop in the final reward. This is because the agent now cannot pass the particle through any slit, so that if the noise pushes the particle away from the designated one, the agent has to act more often.

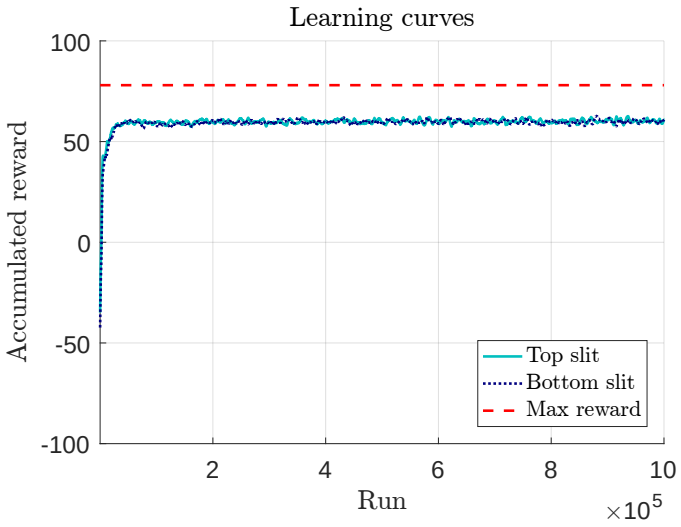
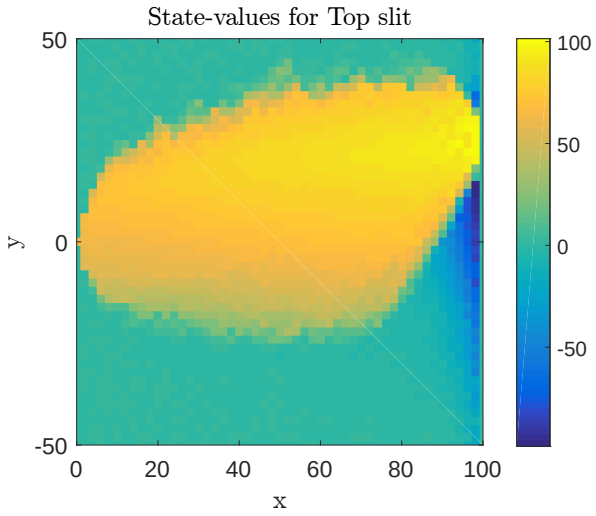


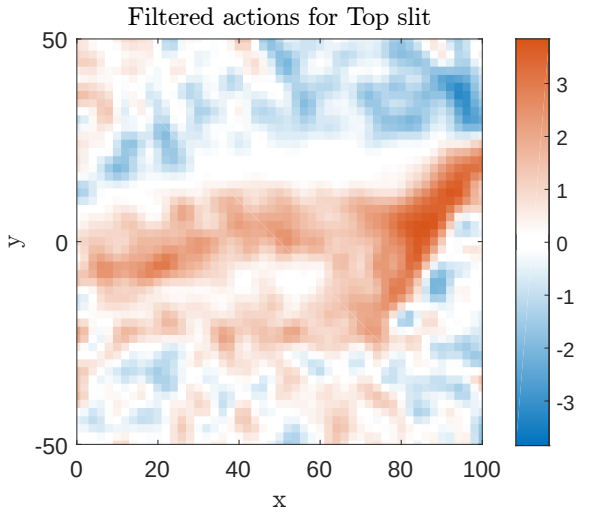
Figure 4.2: Learning curves of the single agent for top and bottom slits (averaged over a moving window of size 5000). The curve are superimposed because the learning is similar in both case as the problem is symmetric.

To assess the two policies that the agent has learned, we can observe the state-value functions and the repartition of actions over the grid in Fig. 4.3. Fig. 4.3a and 4.3c contain the state-value functions. As in the previous scenario, we observe a "spread" of high values from the correct slit. However, this time the other one is ignored and considered as a wall, so that it is not symmetric anymore around $y = 0$. The actions in Fig. 4.3b and 4.3d show that the agent tries to keep the particle on a path that leads to the correct door. This time, the allowed expansion in y is limited, and the particle is more confined. As we will see next, this causes the accumulated reward to be lower than in the multi-goal case.

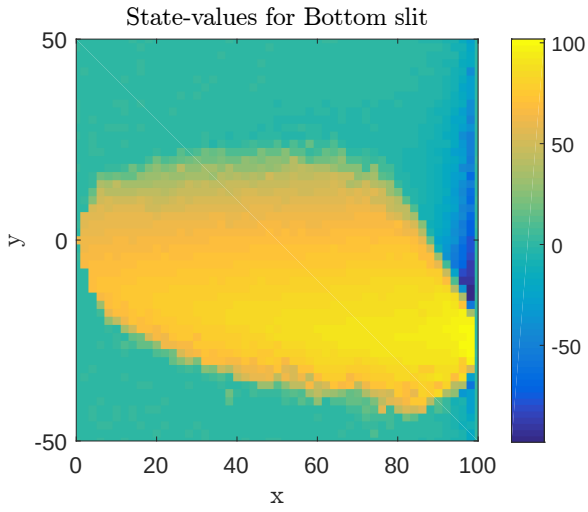
To confirm this, we run once again the problem 200 times with a greedy policy, from which a hundred runs are with the top slit designated, and the other hundred with the bottom one. On the graph (see Fig. 4.4), this is represented by different colours. We are indeed able to observe the two wide "paths" that the particle is forced to take toward the correct slit. Moreover, as stated before, the average reward decreases from the multi-goal case, as the agent has to keep the particle in the correct half of the grid.



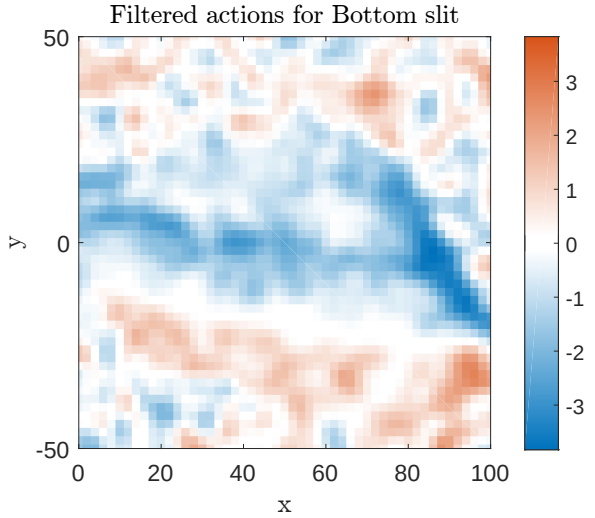
(a) State-values for the top slit scenario. The other one is considered as a wall.



(b) Actions for the top slit, filtered with a 5-by-5 Gaussian filter



(c) State-values for the bottom slit scenario. The other one is considered as a wall.



(d) Actions for the bottom slit, filtered with a 5-by-5 Gaussian filter

Figure 4.3: State-values and actions for the two policies of the single agent, single-goal scenario. Top row corresponds to top slit, and bottom row to bottom slit.

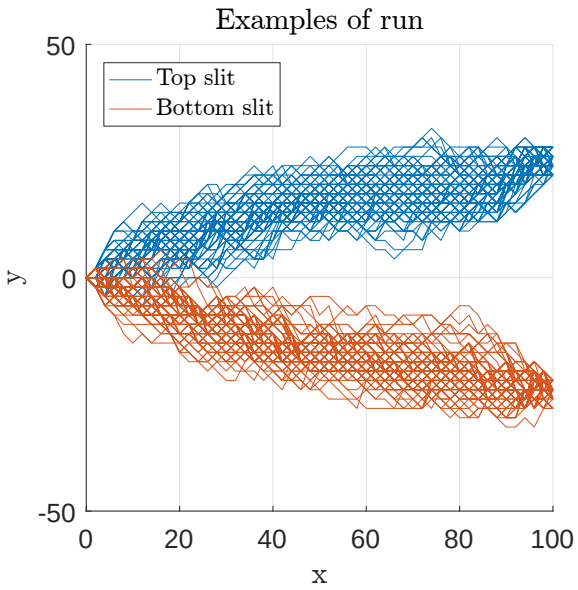


Figure 4.4: Trajectories of 200 runs (100 for each slit choice) after learning by the single agent, in the single goal scenario. The colour indicates which slit has been initially designated. (Average reward, mean $\pm$ std for Top: 69.44 ± 8.40; Bottom: 68.44 ± 8.29)

4.2 Multi-agent

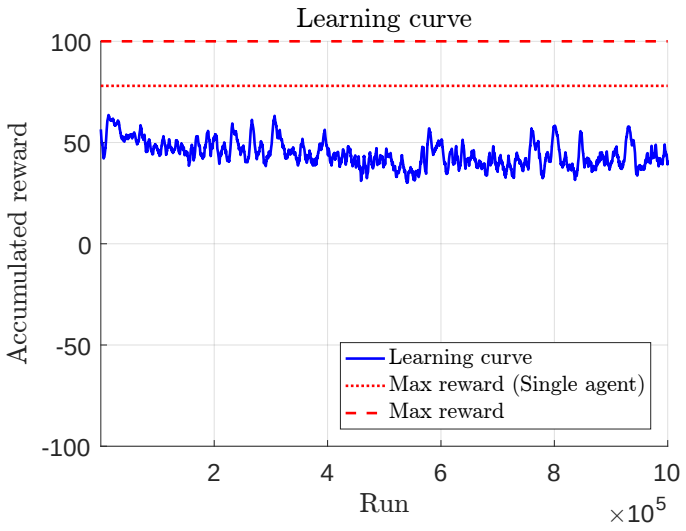
Now, we move onto the multi-agent case: two agents act on the particle. The leader has an intention, a certain goal (he then has two policies, one for each task), and the follower tries to help, but is not aware of the designated door (he has only one policy). To force the follower to learn to assist the leader, we don't give it any costs for its actions, but we rather give it the leader's. That is, they share the same reward function.

4.2.1 Non-adaptive leader

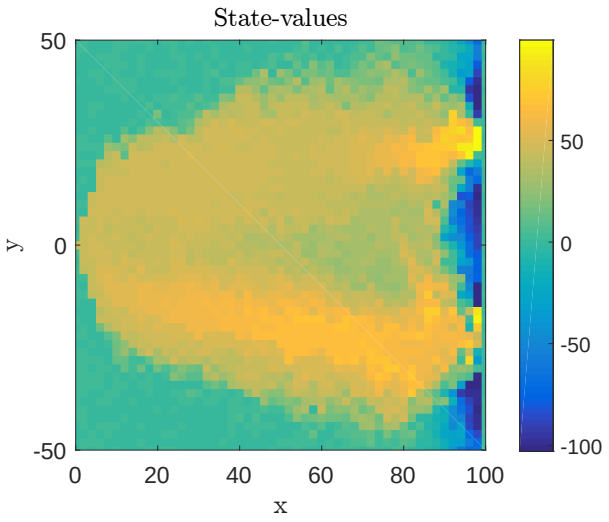
In this first scenario, the leader is "fixed", i.e. it has already learned how to pass the particle through the correct door, and the follower is simply added. Practically, the leader's policies are the one learned in the single-agent/single-goal scenario. This leader won't learn anything while the follower does, it is said to be non-adaptive. During the learning, the slit is randomly designated at the beginning of each run. The resulting learning curve can be found in Fig. 4.5a. The dotted red line serves as a recall of the maximal reward in the single agent case. The dashed line represents the maximal reward that the agents can get: it is simply the reward of the slit with no movement cost if the follower is ideal and is able to guess directly the correct door. As for the learning curve, we observe that the agent already starts with a positive reward. This is because the leader has already learned and knows how to behave to reach the correct slit. However, the learning seems to worsen as the follower learns to act on the grid, we then expect some problem. By looking at the state-values (Fig. 4.5b), we observe that they are in general lower than for the single agent. This means that the follower expects a lower reward on average from those states. This is in accordance with the last paragraph. As for the actions, even though they seems relatively adequate (see Fig. 4.5c) as the follower tries to move the particle through two paths directed toward each slit, the performances seem to deteriorate. Indeed, in Fig. 4.5d we plot the trajectories of 200 runs to see the results of the learning. The colours represents once again the designated slit. We notice that the agents are not always able to get to the correct door (37 times they are wrong). As the difference in reward between slit and wall is high ($+100$ for the slit and -100 for the wall), the average reward declines abruptly as the standard deviation increases.

4.2.2 Adaptive leader

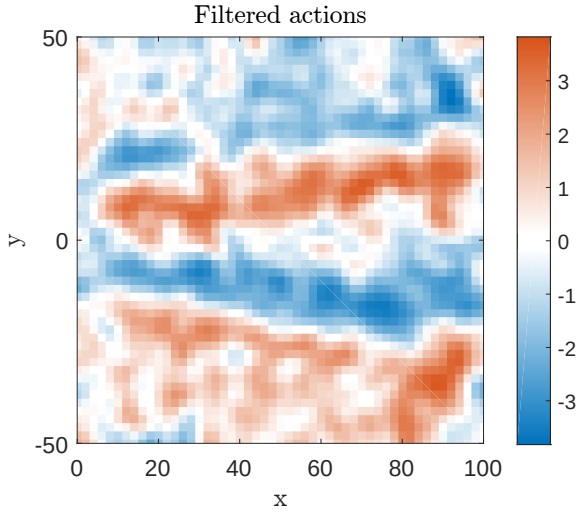
In the final scenario, the leader is now a learning agent like the follower. They both start from zero and will learn together to complete the correct task. To achieve this, the leader will have to learn two policies as before, while the follower learns its unique "assistive" policy. As the leader is now learning, it is said to be adaptive. Here again the slit is randomly designated at the beginning of each run during the learning. As in the single agent case (e.g. Fig. 4.2), the agents quickly learn at the beginning (see Fig. 4.6). However, instead of reaching a plateau where the accumulated reward doesn't increase much, the agents still slowly learn to improve their performances. We can also note that the curve seems more stochastic than in the single agent case. This is because of two reasons: first, the high reward difference between wall and slit causes a high variation between successive runs, and second, now both agents explore, thus there is a higher chance of hitting a wall because of this. Thus, it is possible that a few successive failures (due to exploration) happen so that the averaged value of the accumulated reward drops momentarily. We could also notice that the learning curve seems to reach the expected maximal reward of the single agent case. However, this is not really relevant and might simply be a coincidence coming from the grid and rewards



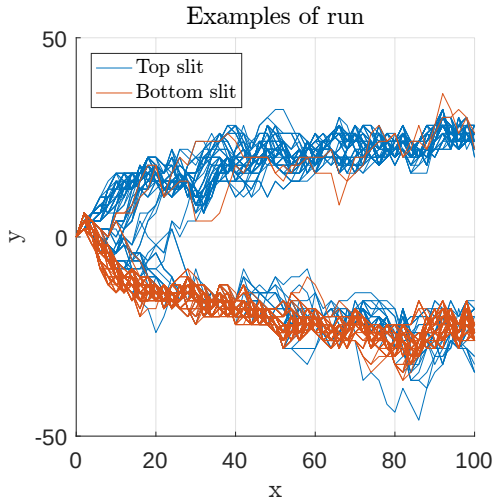
(a) Learning curve of the follower, averaged over a moving window of size 5000. It starts relatively high because the leader already knows the tasks, but doesn't improve.



(b) State-values of the follower. The values are not as high as in the single agent case.



(c) Actions of the follower, averaged with a 5-by-5 Gaussian filter. There are two trajectories that lead to both slits.



(d) Trajectories of 200 runs after the learning by the follower. The colour indicates which slit has been initially designated by the leader: 37 goes to the wrong door. (Average reward, mean ± std: 36.95 ± 97.05)

Figure 4.5: Results of the multi-agent scenario, with a non-adaptive leader.

distributions.

We will now try to understand the policies that the leader has learned. The state-value functions can be found in Fig. 4.7a and 4.7c, with their corresponding actions in Fig. 4.7b and 4.7d. From the state-values, we observe that the grid seems to be divided in two different areas: the states of high values form a region which does not overlap with the one of the other policy. The leader is then trying to keep the particle in the specific region that leads to the designated door. Indeed, on

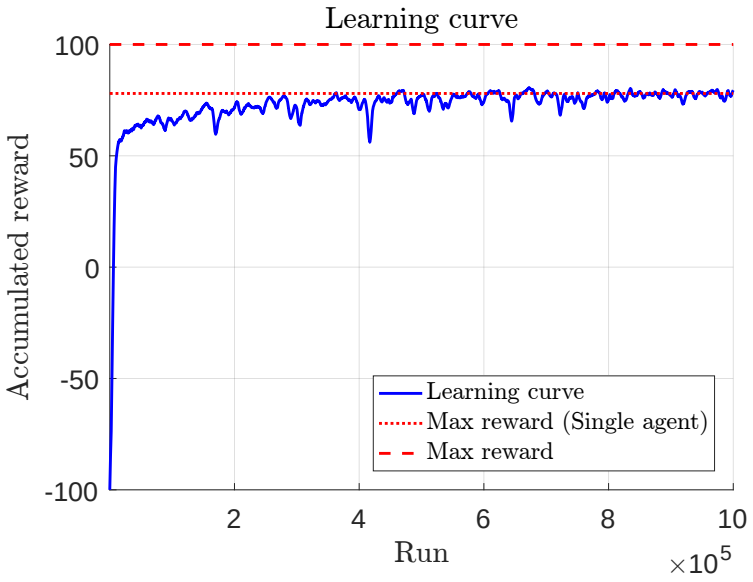
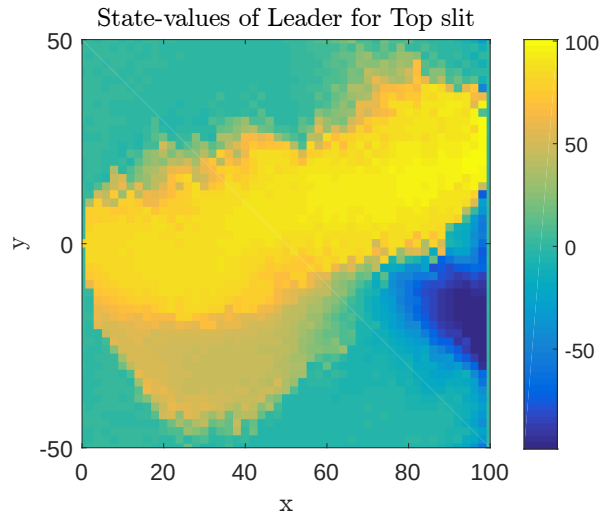


Figure 4.6: Learning curves of both the adaptive leader and the follower (averaged over a moving window of size 5000). As they share the same reward, their curves are identical.

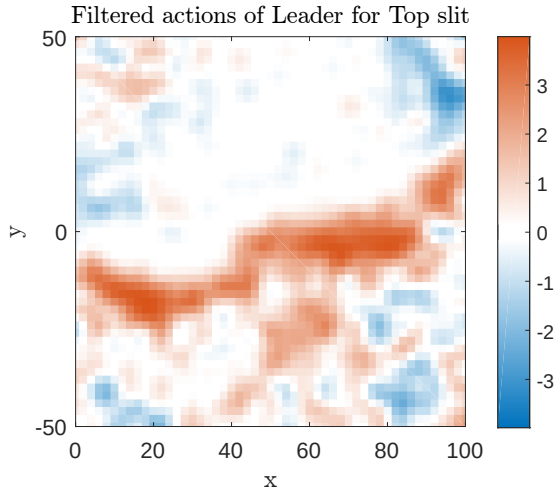
the actions plots we notice that the leader is pushing the particle in these sections of the grid but doesn't try to control it further, except at the end of the grid where it moves it toward the slit.

To finally explain how the agents act together, we interpret the policy learned by the follower. Its state-values can be found in Fig. 4.8a, and its actions in Fig. 4.8b. We observe that the state-values seem to be a combination of the leader's: for the follower both "regions" are good as this policy should work for either door. However, we are able to deduce more interesting behaviour from the actions. It seems that the follower tries to confine the particle into two different trajectories that each leads to a slit. More interestingly, the trajectory that leads to a certain slit is situated in the specific favoured region associated with it in the leader's policies.

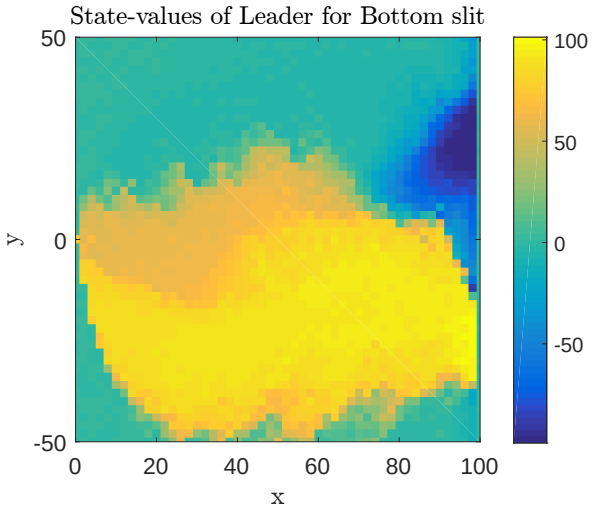
After both agents' learning, we run the problem multiple times and plot the particle's trajectories. We designate the top slit for 100 runs, and the bottom one for another 100. The results can be found in Fig. 4.9, with the colour indicating the initial choice of door. We notice that this time every task is correctly effectuated by the agents. Moreover, we can see the two "paths" that the follower confines the particle into. We observe that these two paths are relatively close to each other, even though they do not overlap apart from the beginning.



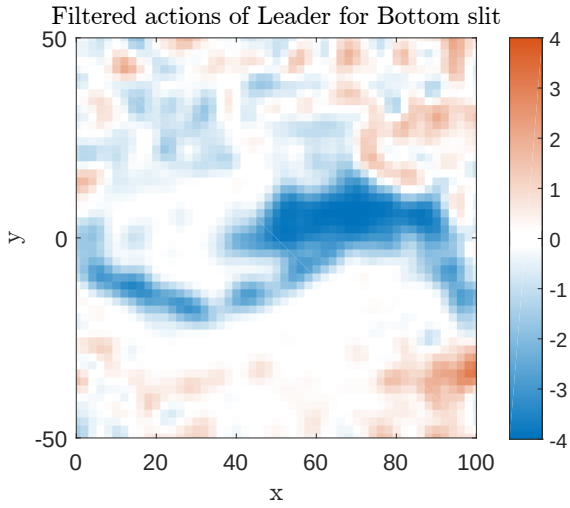
(a) State-values when the top slit is designated as the goal. We observe a first "region".



(b) Actions for the top slit, filtered with a 5-by-5 Gaussian filter. The leader pushes the particle in this region.

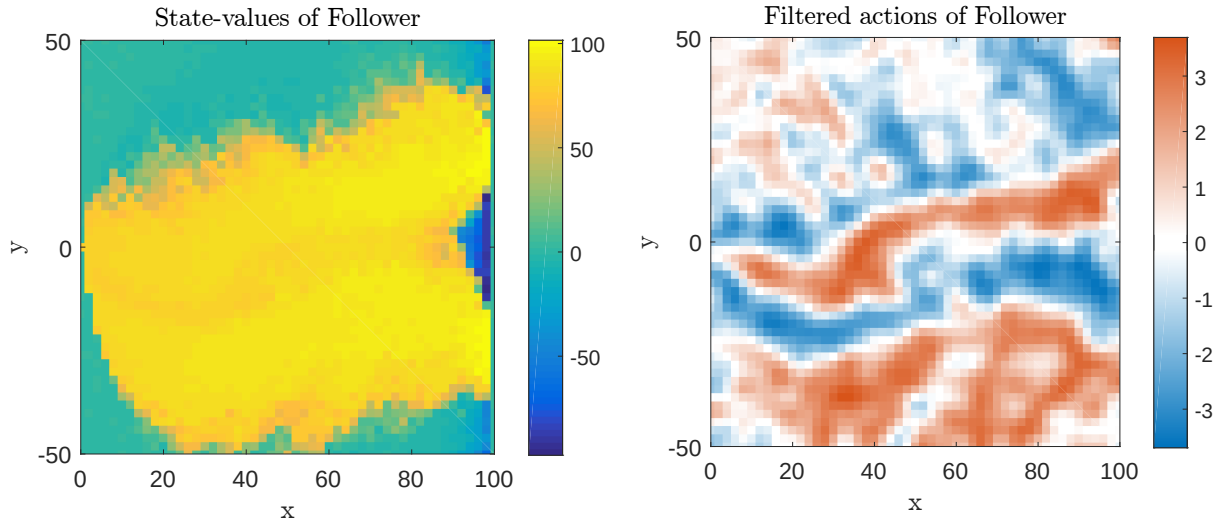


(c) State-values when the bottom slit is designated as the goal. We observe the second "region" that is separated from the first one.



(d) Actions for the bottom slit, filtered with a 5-by-5 Gaussian filter. The particle is also pushing the particle in the second region.

Figure 4.7: State-values and actions of the adaptive leader for the multi-agent scenario. The first row corresponds to the policy of the top slit choice, the bottom row to the policy of the bottom slit.



(a) State-values over the grid. They seem to be a combination of the leader's.

(b) Actions over the grid, filtered by a 5-by-5 Gaussian filter. There are two trajectories that lead to each slit. They are situated in both "regions".

Figure 4.8: State-values and actions of the follower for the multi-agent scenario. In this case, the leader is adaptive.

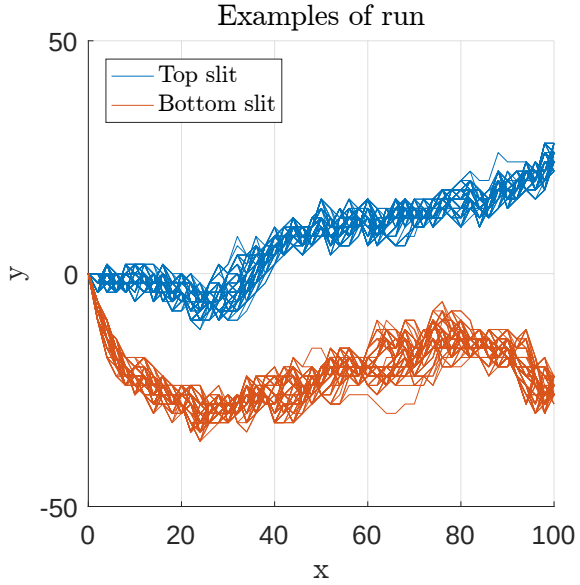
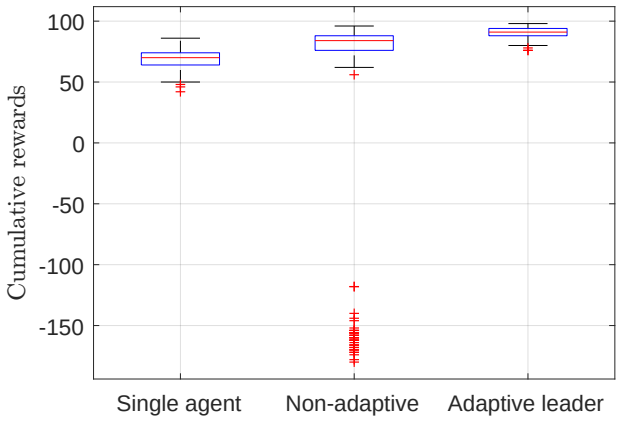


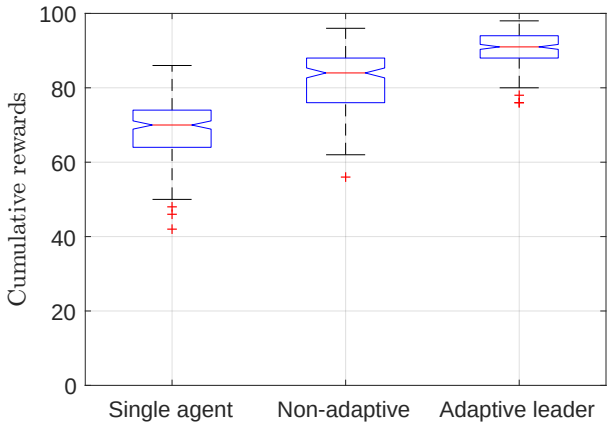
Figure 4.9: Trajectories of 200 runs after learning by both the adaptive leader and the follower. The colour indicates which slit has been initially designated: all tasks are correctly effectuated. (Average reward, mean $\pm$ std for Top: 93.30 ± 3.00; Bottom: 87.52 ± 3.88)

4.3 Performances comparison

We want to evaluate the evolution of performances from single to multi-agent to be able to see if adding a follower in our scenario is actually relevant. To do so, we consider as performance the accumulated reward of the agent(s) for an entire run after learning. We take the results of the 200 runs performed in Fig. 4.4, 4.5d, and 4.9 as the samples for this comparison. As we have seen for the multi-agent with non-adaptive leader (section 4.2.1), the high reward difference between the slit and the wall causes the accumulated reward to either be high near 100, or really low, below -100. Then, the reward distribution is far from being like a normal distribution. This is why we rather use boxplots as a way of comparing them. The boxplots of the single agent (single goal scenario), along with the multi-agent with both non-adaptive and adaptive leader can be found in Fig. 4.10. We can note a clear improvement of the results when we add a follower. This indicates that the agent indeed provide an effective assisting behaviour. However, in the non-adaptive leader case, we observe far outliers (see Fig. 4.10a). Each one of these is actually a run where the wrong door has been reached: despite showing an improvement over the single agent case when the correct task is effectuated, around 15% of the time the follower will actually prevent the leader from accomplishing its goal.



(a) Boxplots of the accumulated rewards obtained over the 200 runs after learning. In the multi-agent case with a non-adaptive leader, we observe 37 outliers with a high negative reward.



(b) Same boxplots with notches added, but with only the positive part shown to better discriminate the performances (even though the non-adaptive's outliers should not be forgotten). As the notches don't overlap we can reject the hypothesis that their medians are similar with 95% confidence. Thus, there is indeed an improvement.

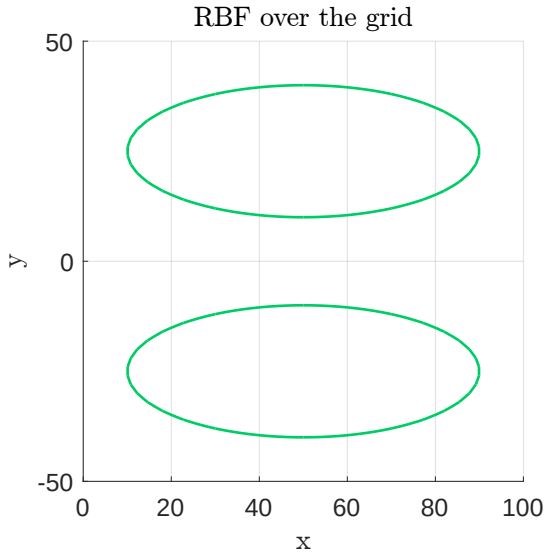
Figure 4.10: Comparison of the performances between single-agent, multi-agent with non-adaptive and adaptive leader. The performance is the accumulated reward that the agents received over 200 runs. Note that for the multi-agent case, the reward corresponds to either agent as they share the same function.

4.4 Extension to continuous state and action

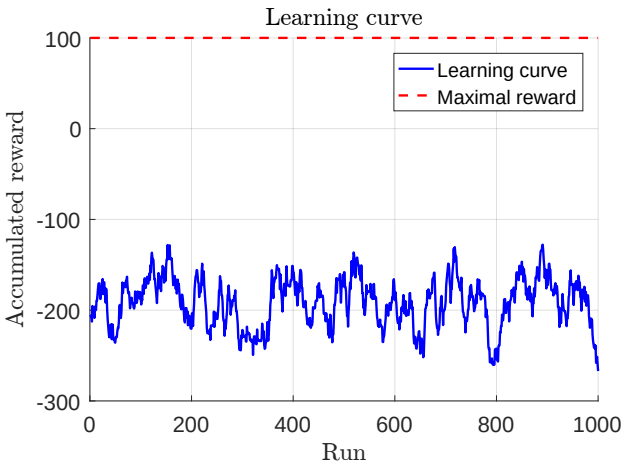
We wanted to start with the most simple case we could think of in order to evaluate if Policy Gradient could be implemented, and could work for our problem. To do so, we have reduced

the double slit problem to something really simple: only one agent tries to make the particle go through a single slit centred in [100; 0]. The feature vector was simply built by putting two Radial Basis Functions, on each side of the slit. We give an illustration of the RBF on the grid in Fig. 4.11a. With this basic setup, the resulting policy can only consist in two horizontal regions (top and bottom) where the agent will perform a certain type of action (up or down), with a gradual transition in between. When looking at the learning curve, plotted in Fig. 4.11b, we are unable to see a nice learning behaviour like in some previous cases (e.g. the single agent in Fig. 4.2). Instead, it seems that the agent does not really improves its reward during the learning, similarly to the non-adaptive leader scenario (Fig. 4.5a). However, as it is only the learning, it is still possible that the exploration is causing this (by making the particle go in a wall), and the agent then needs to be evaluated by other means. For this, we look at the policy it learnt to understand its behaviour, and to see if it would be able to achieve its goal. The policy, that is the action to be done in each state, is plotted in Fig. 4.11c. We can see that it pushes the particle up when in the bottom region, and pushes it down when in the top one. This is actually the wanted behaviour we would want the agent to learn, as it keeps the particle in front of the slit. After this, we assess of the agent's performance by running a hundred runs and plotting the particle trajectories, shown in Fig. 4.11d. We can notice two things. The first one is that the particle indeed stays in a straight trajectory towards the slit. And second, this straight trajectory is actually quite wide (in the y direction). This becomes problematic as the particle approaches the end of the grid, as it can hit the wall instead of passing in the slit. This can explain why the standard deviation of the accumulated reward is high (73.61), despite having positive results overall (the mean is 27.93).

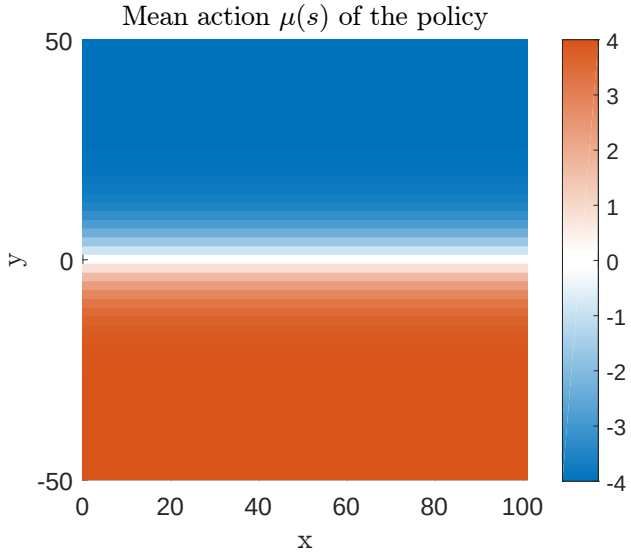
However, we presented here a successful learning. As policy gradient is based on a gradient ascent, it has risks of getting stuck in local optimum. Thus, we expect the results to vary. To confirm this, we perform multiple learnings, and each time we evaluate the agent over a 100 runs. We then compute the mean and standard deviation of the accumulated rewards. The results are given in Fig. 4.12. When the mean rewards is around 0, we fall back in the previous case and can consider the agent to have correctly learned the task (the standard deviation is high for the same reason as before). However, if the mean is far more negative, that would mean that the agent is not able to pass the particle through the slit. We observe that only 4 of the 10 learnings lead to a good policy, while the other 6 caused the reward to be extremely low. Therefore, the policy can indeed get stuck in local optimum, and the results are not constant over different trials. We also need to note that increasing the number of iterations only causes more computations without changing the results found here.



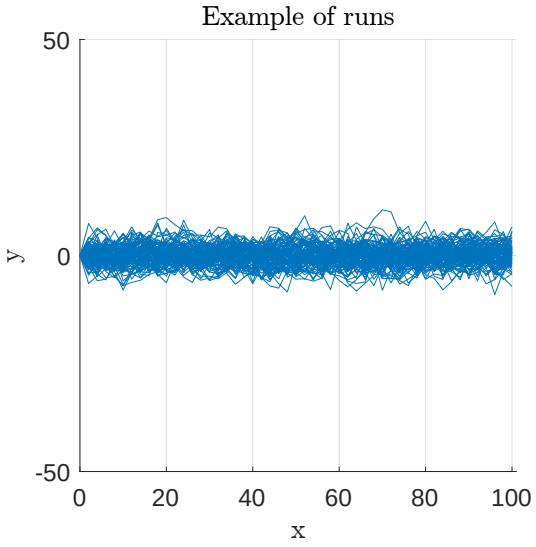
(a) Illustration of the Radial Basis Functions on the grid used for constructing the feature vector $\phi(s)$. The plotted contour correspond to the isoline where the RBF equals $e^{-1/2} \approx 0.6065$.



(b) Learning curve of the agent (averaged with a moving window of size 20). The maximal reward that the agent can get is 100 because the particle starts directly in front of the slit, thus no movement would be required without noise.



(c) Mean $\mu(s)$ of the Gaussian Policy over the grid. It represents the action that the agent has learnt to do in the state s.



(d) Trajectories of 100 runs after learning by the agent (Average reward, mean $\pm$ std: 27.93 ± 73.61).

Figure 4.11: Results of a good learning with Policy Gradient. There is a single slit at [100; 0], and we use a simplified feature vector.

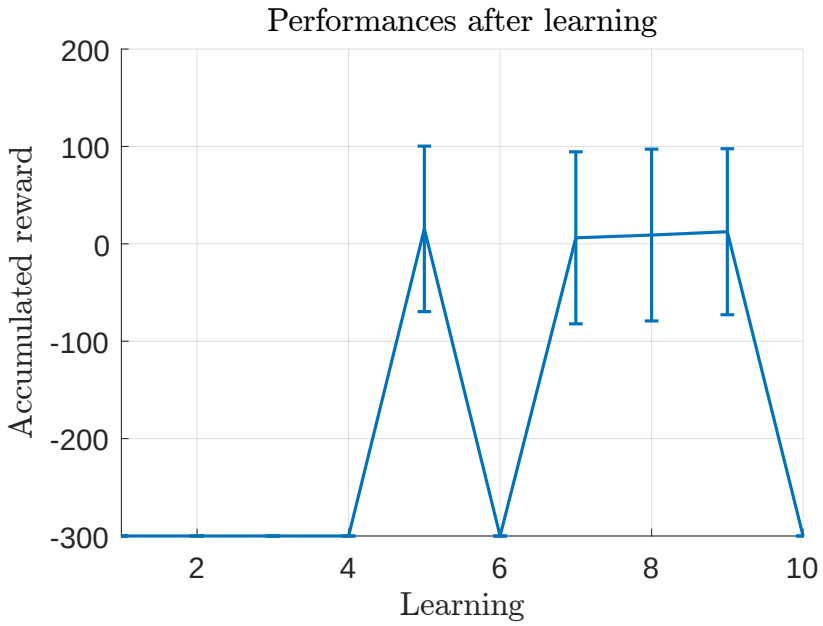


Figure 4.12: Evaluation of performances over 10 different learnings. Each learning was done over 1000 runs. The performances was computed as the mean and standard deviation of the accumulated rewards that the agent would receive on 100 runs after learning.

Chapter 5

Discussion

In this chapter, we discuss the results previously shown to try to understand how the agents have been able to overcome the uncertainties to achieve their goals. As it has been noted in section 4.3, there is an improvement of the performances, that is an increase in the collected reward, on the double slit problem when there are two agents joining their actions. This has an importance as it allows us to see that the follower agent does indeed assist the leader agent instead of preventing him to achieve its goal. Thus, the results suggests that the follower has inferred the intention of the leader, as it seems that they collaborate to pass the particle in the designated slit.

5.1 Learning to lead

We have seen two different ways of joining agents on the grid by taking a non-adaptive and an adaptive leader. In either case, we have been able to see an improvement over the single agent scenario (see Fig. 4.10b). However, we need to consider the outliers (shown in Fig. 4.10a) before asserting that both leaders lead to better results. Indeed, it seems relatively frequent in the non-adaptive leader's case not to reached the correct slit (if we recall the trajectories of the runs in Fig. 4.5d, 15% lead to the wrong slit). Then, we can deduce that this type of leader is actually unreliable. Moreover, we can add that the performances of the adaptive leader are even better that the non-adaptive's. Therefore, we can conclude from this that not only the follower should learn, but that to efficiently overcome the uncertainties, the leader should be learning too. For a human-robot interaction, this would mean that the human should also adapt to its robotic partner in order to be the most efficient. Additionally, if the leader is adapting, the follower should adapt too, or it is going to become obsolete. Thus, we can conclude that it might not be sufficient to develop a control based simply on demonstrations of the tasks by the human, but that the robot should keep updating its behaviour during the interactions.

5.2 Disambiguation is the key

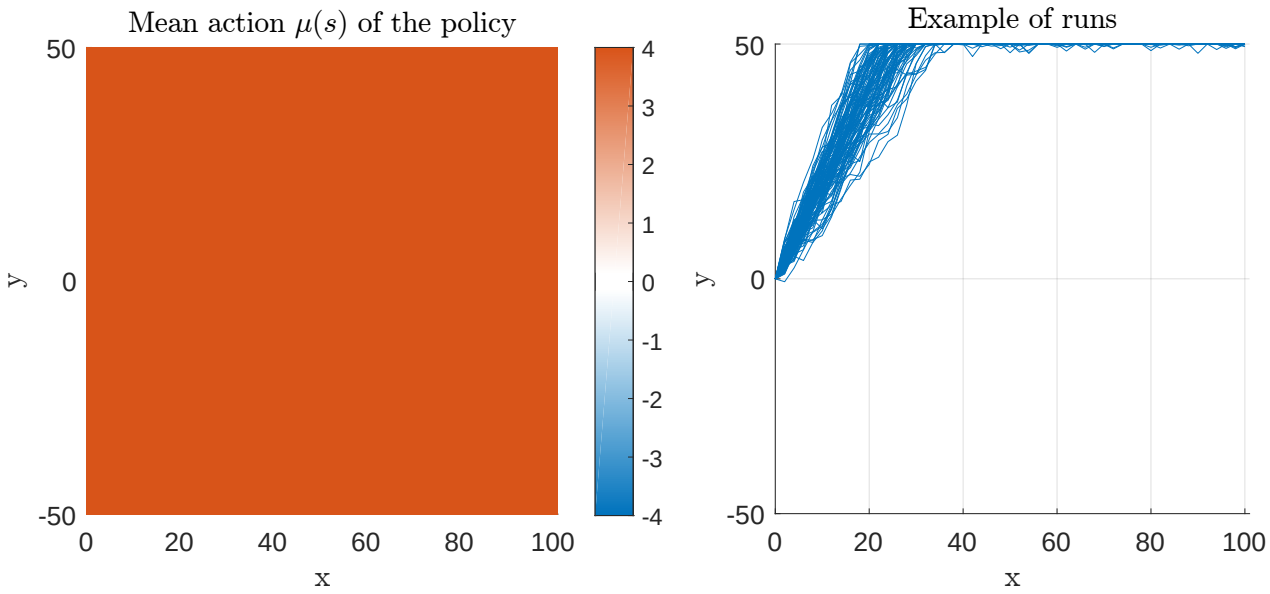
We now analyse the results of the multi-agent cases in order to understand the behaviour of the agents. Starting with the non-adaptive leader, we recall the actions of the follower (Fig. 4.5c), and the results of the learning on multiple runs (Fig. 4.5d). The agent has learned to guide the particle to both doors, but seems to have difficulties to infer his partner's intention as not all trajectories lead to the correct slit. Moreover, we observe that the differentiation happens at the beginning of the grid, where the follower quickly tries to infer the goal, and then keeps the particle on a path toward the corresponding door. However, if it happens that this door is not the designated one, the

leader won't be able to correct the follower that forces the particle on this specific trajectory. Thus, the uncertainties caused an ambiguity between the tasks that the follower might misinterpret, and the leader is unable to correct the wrong guess. When the leader is adaptive, results suggests that the agents have develop a strategy in order to pass the particle through the designated slit. The leader's role is to bring and keep the particle in the part of the grid where the follower is able to understand which slit is correct, and therefore guide the particle toward it (Fig. 4.7 and 4.8). The leader has then learned to disambiguate the task for the follower. Moreover, in between the two trajectories of the follower's action (Fig. 4.8b) there is another "unstable" trajectory in the sense that any small deviation will cause the follower to shift the particle toward one of the two previous paths. Instead of waiting for the leader to entirely push the particle on the correct path, the follower is actively trying to understand the goal. It might also be a way of showing to the leader what it has understood so that it can get corrected quickly if needed. Actually, if we look at the leader's actions (Fig. 4.7b and 4.7d), we notice that it is the most active in this particular region. Therefore, it appears that this area of the grid is where the agents try to disambiguate what they know. Additionally, this region is relatively thin. This can be helpful if the follower needs correction as the leader might be able to overcome the follower's actions in order to get the particle back in the correct part of the grid. We may also add that in our case, the disambiguation of the task comes directly at the beginning (Fig. 4.9) in order to maximize the total reward by minimizing the leader's needed actions. Consequently, these results suggests that to overcome the uncertainties, the disambiguation is key. To help the follower infer its intention, the leader has to clearly disambiguate the task so that the uncertainties along the way are not sufficient for the follower to interpret something wrong. In addition, it seems that the follower should also show clearly to the leader the task it has inferred so that it can be corrected easily and in time in the eventuality that it is not correct.

5.3 From discrete to continuous space representation

As an attempt to generalize our findings, we tried to extend our implementation of the double slit in Reinforcement Learning by going from a discrete world to a continuous one. We initially wanted to show that the problem could indeed be solved using Policy Gradient with Radial Basis Functions for computing the features. As we saw from the most simple case we could think of (Fig. 4.11), it is actually possible to solve this problem with Policy Gradient methods, as the agent could learn a policy to pass the particle in the door. However, we also learned from Fig. 4.12 that it is not always successful, and that the agent might get stuck in local minima. Furthermore, the agent's action are bounded (in our case to $[-4; +4]$), and so are the parameters of the policy. Thus, it is possible that the gradient ascend causes the parameters to get stuck at their boundaries. An example of the results of such badly learnt policy is shown in Fig. 5.1. The actions in Fig. 5.1a are all equals to the maximal vale of $+4$. Then, the accumulated reward is always the worst possible, -300.0 (-100 from the wall, and $-4 * 50 = -200$ for actions taken at each time step), as the particle is always pushed to the top, and eventually hit the wall (see examples of trajectories in Fig. 5.1b). Because of these learning problems, we might even expect the multi-agent case to worsen.

Therefore, a lot more work should be done in order to improve the Policy Gradient implementation to see if we find similar conclusions as in the discrete case. To do so, we would need numerous RBF over the grid to develop more complex policies (Fig. 5.2 shows an example of what it could be). Nevertheless, a higher complexity would only make it harder to learn. Something thus need to be made in order to correct the learning. From the results on Fig. 4.11d, we realise that there



(a) Mean $\mu(s)$ of the Gaussian Poliy over the grid. Every action is $+4$.

(b) Trajectories of 100 runs after learning by the agent (Average reward, mean $\pm$ std: -300.0 ± 0.0). The particle is continuously pushed upwards, and always hit the wall.

Figure 5.1: Results of a bad learning with Policy Gradient (single slit at [100; 0], simplified feature vector with two RBF). The policy's parameters are stuck at their boundaries' value.

is probably a problem in using our Policy Gradient implementation with the current grid and slits. It is possible that a Gaussian policy combined with the noise has more difficulty on the same environment used for SARSA. Hence, to improve the overall learning, we might need to adapt the problem to the continuous case. This would in return means that the comparison between discrete and continuous may become unfair as the environment would differ.

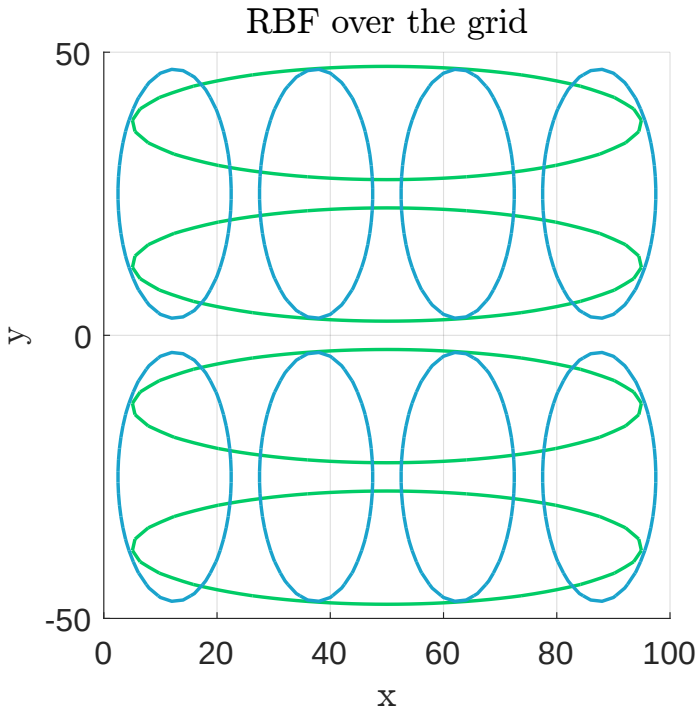


Figure 5.2: Illustration of a combination of Radial Basis Function on the grid to develop more complex policies. The colours serve only to help clarity by differentiating horizontal and vertical RBF.

Chapter 6

Conclusion

Collaboration between humans and robots becomes increasingly present in today's society, requiring an effective coordination between their actions in order to efficiently achieve a common goal. In human-human interaction, some cognitive mechanisms have already been proposed to effectuate these joint actions [1]. For human and robot, the need of inferring the human partner's intention is however more challenging because of the uncertainties.

In this project, we were interested to see how to deduce this intention with the underlying uncertainties. We have used Reinforcement Learning on a simple experiment (i.e., the double slit problem) to assess how two agents were able to accomplish together a given task. The leader agent represented the human, whereas the follower agent illustrated the robotic assistant which needs to guess the leader's objective. We found that a leader which would not adapt its behaviour after introduction of a follower performs worse than an adaptive one. Moreover, it causes the follower to struggle with inferring the task, and eventually leads to some wrong goal predictions. Therefore, peak performances are reached when both partners learn to act together in order to accomplish the correct objective. Additionally, the mechanism that we found central to inference under uncertainties is the disambiguation between the two agents. Thus, the noise was overcome by avoiding states where the task is ambiguous: the leader makes sure that the follower is acting towards the correct task, while the follower acts in order to never stay in the ambiguous states. The uncertainties are then insufficient to change the follower's inference of the leader's intention. After this, in an attempt to generalize the finds we made, we tried to solve the same problem in the continuous case using Policy Gradient method. It was however unsuccessful as the learnings did not give constant satisfactory results, even in the simplest case. We surmise that it is due to the current double slit implementation that might not be well adapted. In addition, as we saw that it is still possible to occasionally learn a working policy, we think that with more time it would possible to adjust the problem in order to achieve acceptable and consistent results.

Consequently, this project could be further developed by validating the finds for different and more general cases. First of all, as we have lastly tried, the setup could be made more complex in order to see if these kind of behaviours keep arising, or it could be extended to more complex tasks. For instance, it might be interesting to try and implement a similar problem with a real robotic arm. It could have to go to two different final points, but does not know which one: the user would move the compliant arm by guiding it towards to correct one. A force sensory feedback could be used as a negative reward for the robot, and the human should aim to minimise its actions, while still having the robot attain the correct position. Ultimately, even though it is closer to cognitive science, it

is possible to try the given problem on humans too. They would have to solve a "mini-game" resembling the double slit, where one would be given the correct door, and the other would have to guess over many tries. They should however not be able to communicate in order to imitates our two agents. Finally, this project has shown the importance of having disambiguation mechanisms for effective collaboration in a task where a human user would be assisted by a robot. Work as already been done to make robots infer humans' intention through an anticipatory control [3], but further researches in this field could be useful to find mechanisms and clues from the human that the robotic partner could easily understand, in order to allow efficient disambiguation.

Bibliography

[1] Sebanz, N, H Bekkering, and G Knoblich. *Joint Action: Bodies and Minds Moving Together.* Trends in Cognitive Sciences 10, no. 2 (February 2006): 70–76. doi:10.1016/j.tics.2005.12.009.

[2] Mutlu, Bilge, Allison Terrell and Chien-Ming Huang. "Coordination Mechanisms in Human-Robot Collaboration." (2013).

[3] Chien-Ming Huang and Bilge Mutlu. 2016. Anticipatory Robot Control for Efficient Human-Robot Collaboration. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction* (HRI '16). IEEE Press, Piscataway, NJ, USA, 83-90.

[4] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Adaptive Computation and Machine Learning. Cambridge, Mass: MIT Press, 1998.

[5] Kappen, Hilbert J. *Path Integrals and Symmetry Breaking for Optimal Control Theory.* Journal of Statistical Mechanics: Theory and Experiment 2005, no. 11 (2005): P11011.