PIR : Mixed-initiative mission

(Bibliography report)

Student1*, Student2*, Advisor1†‡ and Advisor2‡
*Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: {student1,student2}@student.isae-supaero.fr
†Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: advisor1@isae-supaero.fr
‡Direction Générale de l'Armement - Maîtrise de l'Information, 35998 Rennes Armées, FRANCE
Email: {advisor1,advisor2}@intradef.gouv.fr

***Abstract*—The abstract goes here (100 words max).**

I. Context

The increasing autonomy of robots led to make use of them in a more and more large range of operations and activities, for instance in rough places for surveillance or inspection of contaminated sites. Because the machines cannot be held responsible in case of failure and do not possess the human creativity to improvise, those activities require the assistance of humans. In a mission led by an autonomous robot under the surveillance of a human operator or pilot, the latter is often viewed as a providential agent, capable of resolving any problem and responding to a failure of the robot instantly. But many factors can also lead to a failure of the mission, regarding the human intervention, such as a misinterpretation of the information given to the pilot, a poor designed user interface, or the tunnelling of the pilot on a certain task or point of the mission [2]

Regarding the state of the art, the "Fitts list" [1] , which is a compilation of the general strengths and weaknesses of machines and humans, was traditionally used to determine the function allocation between human and machines. In order to determine the level of human versus autonomy involvement, it is important to know the individual strengths of each actor. On the one hand, humans are capable of complex social interactions, moral judgement, operate well in dynamics environments and are highly flexible. On the other hand, the machines are capable of higher computational performance, produce repeatable results, can heande multiple tasks simultaneously and are not prone to fatigue or boredom [7].

A problem still lies in the fact that the operator is considered as an external heal, capable of resolving all the problems encountered by the machine. In fact, the human operator should no be considered like this, but as a part of a team. Considering this fact, the operator should be monitored to consider if he is capable of making accurate decisions, and allow the use of countermeasures (visual stimuli or alarm). While we know all the information about the machine and its states, we know nothing about the states of the human operator. [4] has shown that the use of a Mixed Observability Markov Decision Process (MOMDP) [5] model was relevant to compute a useful policy taking into account the partially observable cognitive state of the operator.

The cognitive ability of the human operator can be inferred via an eye-tracker device [6] , which shows areas where the operator focuses its attention. Also, the operator's heart beat states, observable via an electrocardiogram (ECG), are successful indicators of cognitive load [8] . The ECG can be coupled with an Eye-Tracker (ET) in order to detect if the operator is in a state of tunnelling, and able to take accurate decisions.

The project presented here is a part of a bigger project that consist in the development of an operator monitoring system using an ET and an ECG, because our point of view is that the human operator is not a providential agent [4] , but an autonomous agent, not always reliable, of the team. Our goal is to make that the ensured performance will be better when the operator is working with the robot. We will develop here the development of the reinforcement learning environment (RL) which will allow us to test, improve and optimize the policy of the machine in a task of surch and extinguish (it is a firefighter robot) in order to integrate the human monitoring system to harvest data to improve this system via an internet site (the interface is visible with the figure 1).

II. Theory

Firstly, a little bit of history for RL : the birth of RL came from the encounter in the late 70s between the computational neurosciences (reinforcement of the synaptic weights of neuronal transmission) and the experimental psychology (animal conditionment models : reinforcement of the behavior leading to satisfaction (ex: Pavlov)) with an adequate mathematical environment : dynamic programming.

A link with experimental psychology is the law of effects of Thorndike (1911) [3] : "Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur : those which accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less

likely to occur. The greater the satisfaction or discomfort, the greater the strenghtening or weakening of the bond".

And so, what is Reinforcement Learning? it is an approach, here computational, to decision-making, understanding and goal-directed learning. The individual learns from direct contact with the environment. It works by the definition of the interactions between a learning agent and its environment in terms of states, actions and rewards. The value and value functions are concepts that are key features of RL methods. Value functions are essential for an efficient search in the space of policies.

Continuing on that point, what are the objectives of RL? The objectives of RL are : the automatised acquisition of skills for decision making in a complex and uncertain environment and learning by "experience" a behavioral strategy (named policy) reflecting the failures and success (reinforcements or rewards).

Some examples of achievements of Reinforcement learning are : the production of the best player of backgammon, players of chess, jungling robots, acrobots, etc.

To begin with the theory, we need first to understand what are a Markov chain, a Markov Decision Process (MDP) and dynamic programming [3].

A Markov chain is a dynamic system with discrete time $(x_t)_{t\in\mathbb{N}} \in X$, where X is space of states such as :

$$\mathbb{P}(x_{t+1} = x|x_t, x_{t-1}, ..., x_0) = \mathbb{P}(x_{t+1} = x|x_t)$$

giving the Markov property that all the useful information for the future prediction is in the present state. A Markov chain in X is defined by a initial state x_0 and the probability of transition $p(y|x) = \mathbb{P}(x_{t+1} = y|x_t = x)$.

Next come the MDP, which is define by (X, A, p, r), where :

- X space of states
- A space of actions
- $p(y|x, a)$: probability of transition from a state $x \in X$ to $y \in X$ when the action $a \in A$ is chosen:

$$p(y|x, a) = \mathbb{P}(x_{t+1} = y|x_t = x, a_t = a),$$

- $r(x, a, y)$: reward when passing from x to y using the action a.

We will note here V the value function and π the policy.

Dynamic programming possess two types of iterations : iterations on the values and iterations on the policies. Having only interest on the policies, we will develop here only the part concerning the iterations on the policies. We build a policy sequence, beginning with a given initial policy π_0, and every step k, we evaluate the policy π_k and calculate V^{π_k}, then we improve the policy by calculating π_{k+1} given by V^{π_k} :

$$\pi_{k+1}(x) \in \text{argmax}_{a\in A}[r(x, a) + \gamma \sum_y p(y|x, a)V^{\pi_k}(y)]$$

We stop when $V^{\pi_k} = V^{\pi_{k+1}}$. It's important to note that the serie $(V^{\pi_k})_k$ is a increasing serie. Therefore, because there is a finite number of possible policies, the termination criteria is obligatory satisfied with k. So we have $V^{\pi_k} = V^*$ where V^* is the optimal value function, therefor π_k is the optimal policy. Some methods of solving are :

- **Direct solving** of the linear system $(I - \gamma P^\pi)V^\pi = r^\pi$. That's the Gauss elimination method, but it have a complexity of $O(N^3)$
- **Iteration on the values for a permanent policy** : we iterate on the operator T^π (Bellman operator). Considering a given value function V_0, $V_{n+1} = T^\pi V_n$. We have then convergence of V_n toward V^π. The problem is that the convergence is asymptotical, but the advantage is that it as a lower complexity that the direct solving method ($O(N^2 \frac{log(1/\epsilon)}{log(1/\gamma)})$ for an approximation of ϵ (interesting if γ is not too close from 1)).
- **Monte-Carlo** : we simulate n trajectories $((x_t^i)_{t\geqslant 0})_{1\leqslant i\leqslant n}$, starting from x and following the policy π : x_{t+1}^i $p(.|x_t^i, \pi(x_t^i))$, so :

$$V^\pi(x) \approx \frac{1}{n} \sum_{i=1}^{n} \sum_{t\geqslant 0} \gamma^t r(x_t^i, \pi(x_t^i)).$$

  this method is interesting in we want to evaluate a unique state. It has an approximation error of the order of $O(1/\sqrt{n})$
- **Temporal differences TD(λ)** : This is a smart method for using the trajectories to evaluate all the states crossed by those trajectories, by evaluating the value of a state x, by the sum of the observed temporal differences $r_t + \gamma V(x_{t+1}) - V(x_t)$ at the future instants t, ponderate by a "trace" λ. The algorithm of TD(λ) will be treated later.

During the experiments (part Experiments (III)), we used some algorithm (Random method, Cross Enthropy Method (CEM), REINFORCE and TD(λ)). We will focus here on the TD(λ) method, which we found giving the most reliable results during the experiments. It is important to note that the Random method was used to test the functionment of the environment, and the debugging of it.

Temporal Differences TD(λ)

the algorithm TD(λ) [Sutton, 1988] : After the observation of a trajectory $(x_0, x_1, ..., x_K = 0)$, we update V_n to the states $(x_k)_0 \leqslant k < K$ following :

$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) \sum_{l=k}^{K-1} \lambda^{l-k} d_l,$$
$$\text{where } d_l = r^\pi(x_l) + V_n(x_{l+1}) - V_n(x_l).$$

The impact of the temporal differences of future transitions on the estimation of the current state value is updated by λ. TD(λ) is a compromise between :

- **TD(0)** (to estimate the fixed point of the operator of Bellman T^π):

$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) d_k.$$

- **TD(1)** (to estimate the mean):

$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) \sum_{l>=k} d_l.$$

implementation of TD(λ) :

With the eligibility trace $z_n \in \mathbb{R}^N$. Once the transition form x_k to x_{k+1} observated, we can calculate de temporal difference $d_k = r^\pi(x_k) + V_n(x_{x+1}) - V_n(x_k)$ and update the eligibility trace :

$$\begin{cases} \lambda_{z_{n-1}}(x) \text{ if } x \neq x_k \\ 1 + \lambda_{z_{n-1}}(x) \text{ if } x = x_k \\ 0 \text{ if} x_k = 0 \end{cases}$$

and we iterate, for every x,

$$V_{n+1}(x) = V_n(x) + \eta_n(x) z_n(x) d_n.$$

the choice of λ :

- $\lambda < 1$ allow to reduce the variance of the estemators regarding $\lambda = 1$.
- $\lambda > 0$ allow to propagate faster the rewards regarding $\lambda = 0$.

the algorithm TD(λ) in the actualised case :

$$V_{n+1}(x_k) = V_n(x_k) + \eta_n(x_k) \sum_{t \geqslant k} (\gamma\lambda)^{l-k} d_l.$$

Here was a little introduction to the theory of reinforcement learning and a little zoom on the TD(λ) algorithm that we used during the experimentations of our environment.

III. Experiment

We now arrive at the core of this project, which was to develop a learning environment describing the firefighting task in order to improve the policy of the robot and improve its autonomy. For this purpose, we used the toolkit OpenAI Gym, which is a toolkit for developing and comparing reinforcement learning algoriths. We also used the open resource which you can find in the github of torch-twrl (you can follow the link : https://github.com/twitter/torch-twrl). With this resource, you can run already existing environments such as a cartpole environment, an acrobot, board games, etc.

We used those resources to implement and develop our own environment which was the firefighting task.

The first task developed was the most simple one : no moving robot, the only task was to detect if a target was on fire, and then proceed to extinguish it.

This environment was first developed as determinist. It was an environment where we knew when and where the target will fire up, and see if the robot was able to learn and take measures against. It's important to note that we used the agent/algorithm cited earlier : TD(λ), REINFORCE, Random agent and CEM.

HERE A GRAPH OF THE AGENTS REWARD

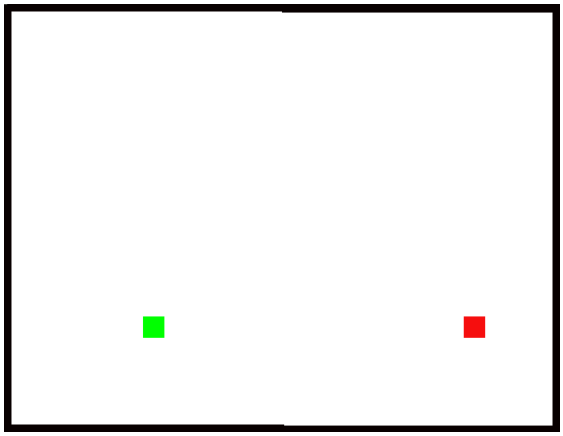

Fig. 1. View of the simulation of the simple determinist environment

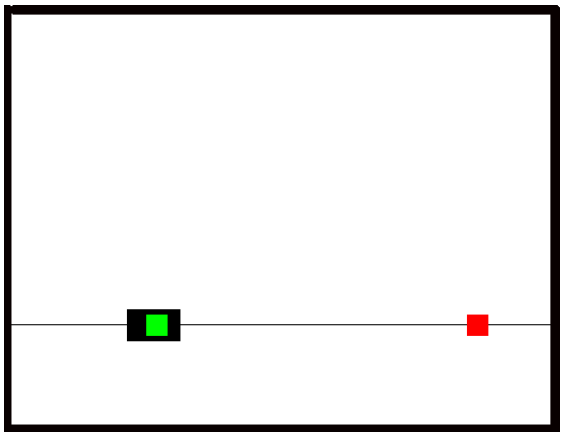

Fig. 2. View of the simulation of the 1D environment

The next step was to turn the determinist environment into a random one, to see if the agents were able to adapt.

HERE A GRAPH FOR RANDOM ENVIRONMENT

After studying the adaptability of the agents to the simplest environment possible for our task, the goal was to develop an environment where there was not the only action of extinguishing the target, but also moving to it.

The first step was still a determinist environment, where we knew when and where the targets will light up.

Results of the 1D environment

At this point, the TD(λ) agent has proven to be the most effective agent.

The second step was to turn the derteminist environment to a random one.

The final step was to develop the 2D environment. The first 2D-environment developed was a simple one, with only targets and the robots. It was used to see if the added actions

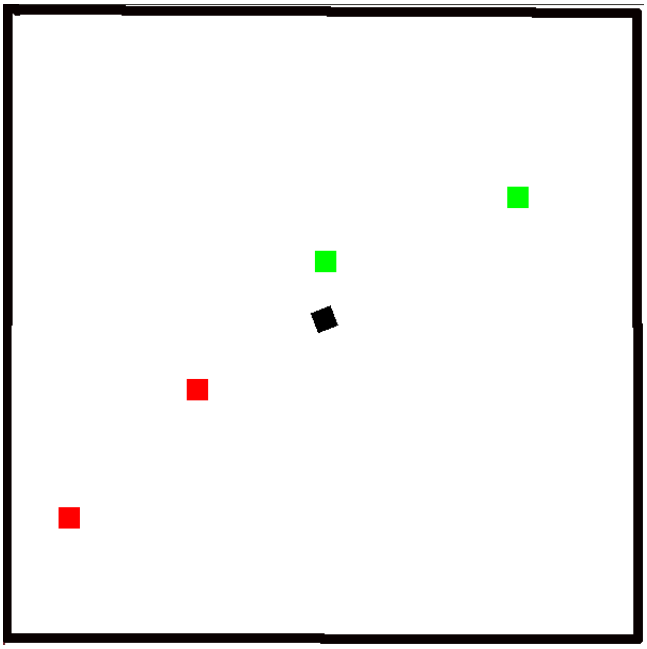

Fig. 3. View of the simulation of the 2D environment

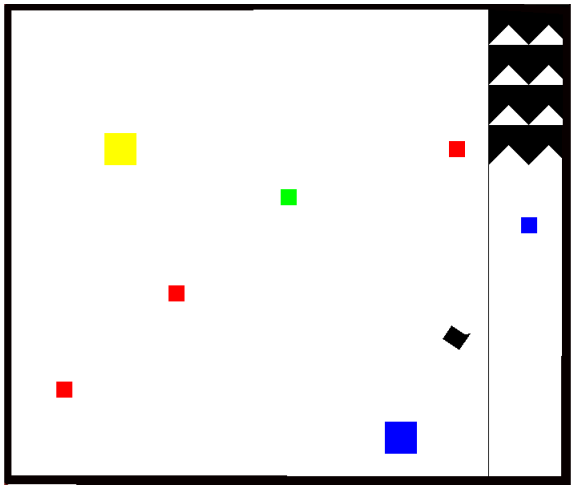

Fig. 4. View of the simulation of the complexified 2D environment

where working accordingly to the real robot.

Then, we developed a more complex environment to make it as close as possible as reality.

To ensure that the environment was close to reality, we discussed the different drawbacks that we needed to implement :

- **the battery** : the energy available for the robot is not infinite, for this, a decrementing counter was implemented, which was decrementing at every action carried out by the robot. There was also a battery zone where the robot could refill up its energy. The battery zone is modelized by the yellow square.
- **the robot's water reserve** : as for the battery, the water that the robot can carry is not unlimited. The water was implemented as a counter which was decremented every time the agent used the action to spill water. This water could be refilled in the water zone which was represented as a blue square.
- **the refilling water zone** : represented as a blue square, the water refilling zone could see its water level decreasing (which was viewed by the fading blue color) due to water leaks (which are part of the operator actions.). Those leaks
- (the human operator) : the human operator needed to be modelized, as for first, he was designed to not take actions directly on the robot, it was designed to act against the water leaks and fill up the water zone. The operator was designed as a random agent.
- **the temperature** : The temperature needed also to be modelized, as the robot could not stay to long close to a fire. It was modelized as a counter that was increasing when the robot was in a range from the fire, and decreasing when it was out of this range.
- **the fire** : The fires modelized in the environment where fires that could be extinguished with only one application of water. We discussed the fact that we needed more than one action to take care of it, but it was not implemented ultimately. But the idea still lies.

You can find the environments and videos of the simulations following this address : https://github.com/TVagneron/Reinforcement-learning

Problems

Some problem occured, and this section is here to describe some of them and maybe help to resolve them.

One of the first issue that we encountered when developping our environment was the definition of the observation space, where we needed to define the observables that can be seen by the agent, as for this we defined the states of the targets by a strict binay duo, 0 when the target was extinguished, 1 if it was lighted. To follow strictly our mathematical model, we used a Tuple to define the target states, but Tuple where not accepted for observation space, so we needed to define them as array and boxes.

The second issue was that when we runned our environment, there was an error ocurring, stating that the local state value was nil. It appears that it was related to a process unable to provide the state at the proper time, resulting in a nil value. This issue was solved by slowing down the process (adding prints to track the error origin solved the proble unexpectedly, but it was also slowed down by adding a wait function in the qfunction). This issue has been observed on two PCs using Ubuntu 14.04. (You can find the issue report following : https://github.com/twitter/torch-twrl/issues/37).

REFERENCES

[1] P. M. Fitts. Human engineering for an effective air-navigation and traffic-control system.

[2] L.Dargent. Profilage pilote par apprentissage automatique pour reconfiguration ihm.

[3] R. Munos. Introduction à l'apprentissage par renforcement.

[4] F. Dehais P. E. U. Souza, C. P. Carvalho Chanel. Momdp-based target search mission taking into account the human operator's cognitive state.

[5] D. Hsu Wee Sun Lee S. C. W. Ong, Shao Wei Png. The human should be part of the control loop?

[6] Mai-Hui Le F. Dehais T. Gateau, C. P. Carvalho Chanel. Considering human's non-deterministic behavior and his availability state when designing a collaborative human-robots system.

[7] S. A. Burden M. K. McCourt J. Williard Curtis W. D. Nothwang, R. M. Robinson. The human should be part of the control loop?

[8] M. M. Wilson. Development of online cardiac feedback for control applications.