Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2022, 9(12):21-31

Research Article ISSN: 2394 - 658X

Comparison between a centralized and decentralized method for multi-agent collaboration in an exploration problem

Giresse Bernard SILEDJE HYOUBISSE*, Leandre NNEME NNEME and Felix Paune

Laboratory of Computer Science Engineering and Automation, University of Douala,
P. O. Box 1872 Douala, Cameroon
*Corresponding author: gsiledje@gmail.com

ABSTRACT

In this paper, two methods for robot collaboration in a multi-agent system (MAS) exploration problem based on reinforcement learning are presented. In this problem, the agents are placed in an arena where a target is located and the goal is to measure the time taken by the robots to detect and destroy it. The experiment was carried out several times in the Robotarium's Matlab API (Application Programming Interface) in order to compare the times taken by each method. In the first method, called centralised, the agents move in groups and traverse the arena from one end to the other, whereas in the second, called decentralised, each agent moves autonomously. The experimental results show that both approaches, distributed and centralized, can finally solve the problem, but the coordination performance of the proposed centralized approach is much better than that of the decentralized approach. We demonstrate through all these experiments that for exploration in an arena with target localization, centralized methods are more efficient because they take less time than decentralized methods.

Key words: reinforcement learning, Robotarium, SMA, exploration, Matlab

---

INTRODUCTION

Swarm robotics is a branch of robotics applying distributed intelligence methods to multi-robot systems. It seeks to study the design and behaviour of robots [1]. Relatively simple rules can give rise to a complex set of swarm behaviours [2]. The cooperation of robots in a swarm already has many applications. [3]. robotic testbeds [4]-[6]. Within a few years it will have a powerful impact in several areas[7] [8] In the army, exploration and rescue missions will be greatly improved thanks to cooperative drones; In home automation, thanks to this system, swarms of cleaning robots could be created [9]In medicine, swarms of robots could explore the human body in search of cancerous cells [10].

The problem posed is that of a set of agents having a common goal, having no idea how to reach this goal, and having to learn the behaviour to adopt to solve this problem. We are interested in situations where agents cooperate. We should therefore find out how agents can learn together to cooperate. This learning is done through artificial intelligence and more precisely through reinforcement learning [11], [12]. This mixture of multi-agent systems and reinforcement learning should allow complex problems to be solved with fairly simple agents while adapting. The aim here is to combine the advantages of these two tools to make a flexible and efficient system.

In an ADM, agents can be confronted with several problems depending on the desired behaviour. For example, one of the major applications of swarm robotics is the search for targets in the environment. This task can have many variants, depending on the nature of the target and the way it is located[13]. Exploration involves the need for a swarm of robots to discover the environment. An example where the allocation of exploration areas is done using the auction coordination paradigm is presented in [14].

Faced with the problems of cooperation and collaboration of robots in a swarm for the realization of a common global task, the main objective of this article is to bring a clear contribution in the SMA. The main objective of this article is to make a clear contribution to the ADMs by proposing in a first step two methods, one centralized and another decentralized. In the second part of the paper, a comparison of the time taken by the robots in each of the methods to locate a target is made. This article is divided into two main parts: in the first part, we present the

methods and tools that allowed us to carry out this work and finally, in the second part we present and discuss the results obtained.

METHODS AND TOOLS

In this section, we present all the different methods and tools that enabled us to carry out our experiments.

Reinforcement learning

In artificial intelligence, more precisely in machine learning, reinforcement learning consists, for an autonomous agent (e.g. robot), in learning the actions to take, from experiments, so as to optimize a quantitative reward over time [12], [15]. The agent is immersed in an environment and makes decisions based on its current state. In return, the environment provides the agent with a reward, which can be positive or negative. The agent seeks, through iterated experiments, an optimal decision-making behaviour (called strategy or policy, and which is a function associating the action to be performed with the current state), in the sense that it maximises the sum of the rewards over time.

Reinforcement learning refers to a class of machine learning problems, the aim of which is to learn from successive experiments what to do in order to find the best solution.

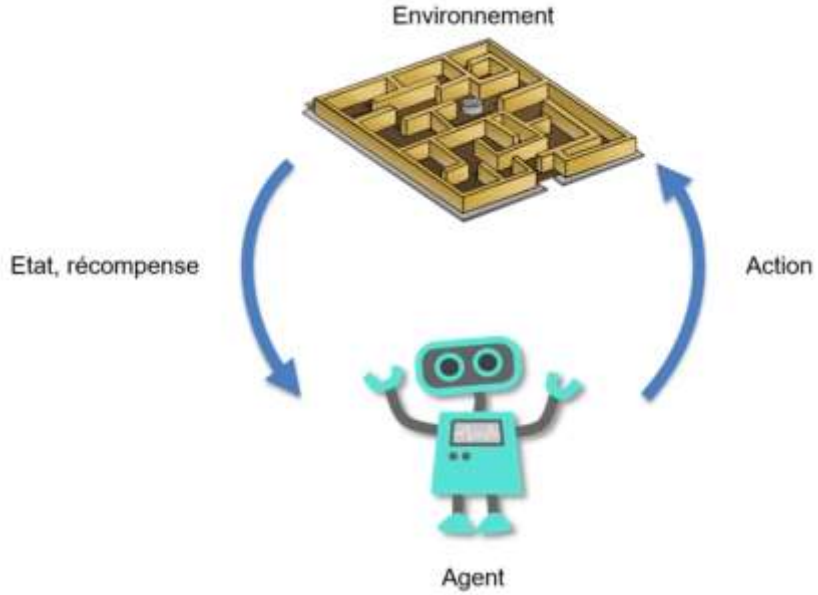

Fig. 1 Representation of reinforcement learning [16]

The Bellman equation

It is necessary to introduce the concepts observed above in order to better implement the Bellman equation. We have a state s, which represents the state in which the environment is. A represents the actions that the agent can take; these depend on the state s of the environment. R represents the reward the agent gets for entering a state. γ =[0,1] which is a reduction factor used to indicate the impact of future rewards on the current expected performance state.

The Bellman equation is defined as [17] :

$$V(s) = \max_a (R(s,a) + \gamma V(s'))$$

This is the next state.

Consider an agent in a maze trying to find the exit. If he arrives at the green point, he receives a +1 reward, whereas if he happens to fall on the red square, he receives a negative -1 reward. The maze is illustrated in the following figure

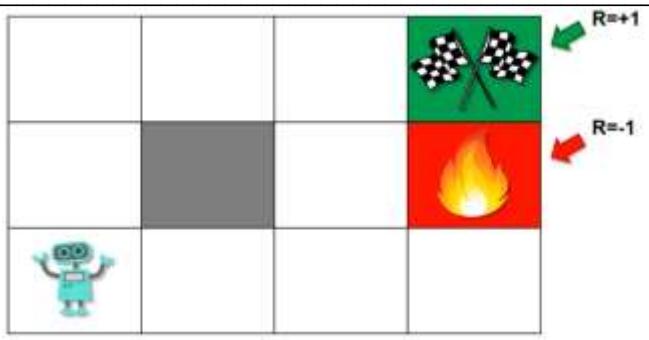

Fig. 2 Illustration of the maze with the agent and the different rewards [16]

Each box here represents a state of the environment. The Bellman equation is used to find the optimal path for the robot to exit the maze. There are two possible cases: a deterministic case where the agent can only go in one direction, in other words, the probability that its direction will change after making a choice is zero. A second stochastic case in which the probability that the agent will not end up in the chosen direction is not zero. This second case will be the subject of our study, as it takes into account unpredictable and uncertain factors that will influence the decision taken by the agent.

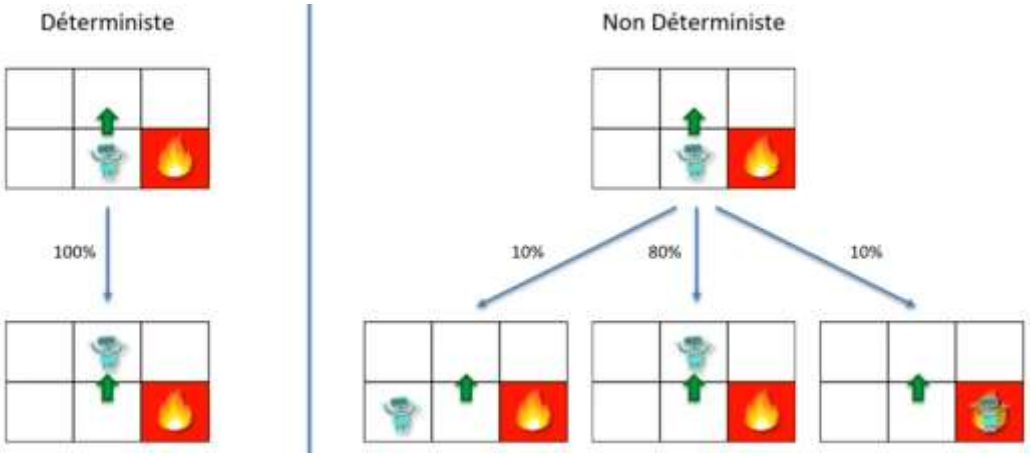

Fig. 3 Illustration of the deterministic and stochastic cases for the agent's choice of direction [16]

This figure illustrates perfectly the two possible cases when the agent makes a choice. On the left, the agent decides to go upwards and has a 100% chance of being there. On the right, the agent makes the same choice, but has a 10% chance of ending up on the right or the left, and an 80% chance of actually ending up at the top. This is therefore a non-deterministic or stochastic case.

This introduces two new concepts, namely Markov processes and Markov decision processes.

Markov processes

A stochastic process respects the Markov property if and only if the conditional probability distribution of future states, given past states and the present state, depends only on the present state and not on past states [18]. This is said to be memory-free. In other words, the probabilities of future states do not depend on past states but only on the present state.

The stochastic process $(X_n)_{n\geq 0}$ with values in a finite set E is a Markov chain if $\forall n \in \square$, $\forall x_0,...,x_n, y \in E$,

$$P(X_{n+1} = y \mid X_0 = x_0,..., X_n = x_n) = P(X_{n+1} = y \mid X_n = x_n)$$

Markov decision processes

A Markov decision process (MDP) is a stochastic model where an agent makes decisions and the results of its actions are random [17]. MDPs are an extension of Markov processes. The difference is the sum of the actions chosen by the agent and the rewards earned by him.

If there is only one action to be taken in each state and the rewards are equal the Markov decision process is a Markov chain.

A CDM is a quadruplet <S, A,T,R> defining

- S is a finite set of states s.
- A is a finite set of actions a.
- T : S × A × S → [0, 1] is a Markov transition function giving the probability of going from state s to state s'when action a is performed P(s,a,s').

- R : S × A → R is a reward function. R(s, a) is the reward obtained by the agent when it performs action a from state s.

The Bellman equation seen earlier thus becomes [18]

$$V(s) = \max_a (R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s'))$$

The Q-Learning algorithm

In artificial intelligence, more precisely in machine learning, Q-Learning is a reinforcement learning technique. This technique does not require any initial model of the environment. The letter Q stands for the function that measures the quality of an action performed in a given state of the system [19]. This learning method learns a strategy, which indicates which action to perform in each state of the system. It works by learning a state-action value function Q which determines the potential gain, i.e. the long-term reward, Q(s,a) , of choosing a certain action a in a certain state s by following an optimal policy [20]. When this action-state value function is known or learned by the agent, the optimal strategy can be constructed by selecting the maximum value action for each state, i.e. by selecting the action a that maximises the value Q(s,a) when the agent is in state s. The mathematical relation of this equation is obtained in the previous section

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s'))$$

Replacing $V(s')$ by its expression, we obtain the mathematical relation of the Q-Learning algorithm

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (P(s,a,s') \max_{a'} Q(s',a'))$$

The Robotarium

The Robotarium, set up by the Georgia Tech Institute of Robotics and Intelligent Machines in the United States of America, is a swarm robotics research platform that is remotely accessible to all. It allows the remote implementation of algorithms on real physical robots in order to make about fifty robots collaborate as illustrated in the following figure:

Fig. 4 Example of a coverage control algorithm executed on the Robotarium using 13 GRITSBot robots. The desired density function is projected onto the testbed arena in the shape of the letter R [4]

The Robotarium simulation API is available in Matlab and Python. The objective of the simulator is to allow users to quickly prototype their distributed control algorithms and receive feedback on the feasibility of their implementation before sending them to be executed by the Robotarium robots.

Dynamic model of Grisbot robots [5]

Consider a differential-drive robot in a global reference frame O with full-state $X = \begin{bmatrix} x & y & \theta \end{bmatrix}^T$ .Let $X_p = \begin{bmatrix} x & y \end{bmatrix}^T$ represent the global position of the robot. Then, consider the following output of the state defined by

$$S(X) = X_p + l \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix},$$

Where $0 < l \in \mathbf{R}$. Geometrically, $S(.) \in \mathbf{R}^2$ represents a point orthogonal to the wheel axis of the robot along the perpendicular bisector of the axis at a distance l . the body velocity of the robot in the global frame can be modelled through the unicycle model

$$\dot{X} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where θ is the orientation of the robot, v and ω are its linear and angular velocity, respectively. Differentiating (S1) with respect to time yields

$$\dot{S}(X) = \dot{X}_P + l\dot{\theta} \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$

Substituting the unicycle dynamic model for $\dot{X}_p$ and io provides the desired relation

$$\dot{S}(X) = R_l(\theta) \begin{bmatrix} v \\ \omega \end{bmatrix},$$

Where

$$R_l(\theta) = \begin{bmatrix} \cos(\theta) & -l\sin(\theta) \\ \sin(\theta) & l\cos(\theta) \end{bmatrix}$$

RESULTS AND DISCUSSIONS

The aim of the game is to program the robots to find and destroy a target hidden in the environment. We have a group of 12 Grisbots, each measuring 10cm x 10cm. They can move and communicate with each other within a certain radius of perception of 30cm.

Scenario

This is the scenario of the experiment. The 12 robots are placed in the middle of a 3.2 x 2 metre arena. A 5 cm target is hidden somewhere randomly in the arena. The robots do not know the location of the target and have to find and destroy it as quickly as possible. A robot can find the target in two different ways: Either it touches the target by moving over it, or another robot that knows the location of the target communicates the information to it (but for this to happen it must be within its perception radius).

When a robot discovers the location of the target, it turns on its green LED.

The target initially has an energy of 100 points. If the target's energy is reduced to 0, the mission is accomplished. To attack the target, a robot must: know the location of the target, have the target within its perception radius. If these two conditions are met, the robot attacks the target automatically. When a robot attacks the target, the target loses 0.3 energy points per second. This is not much, but the attacks of the robots add up. For example, if 10 robots attack at the same time, the target loses 10 times 0.3 energy points per second - that is 3 points/second. A grouped attack is therefore more effective than an isolated attack.

When a robot is attacking the target, it turns on its red LED.

There are many solutions to perform this experiment, in this article we will implement two algorithms (centralized and decentralized). This experiment is carried out in the Matlab API of Robotarim available for free at https://github.com/robotarium/robotarium-matlab-simulator.

The simulation parameters are contained in the table below

Table -1 Simulation parameters

Type	Value	Comment
Time step	0.033	
Robot speed	0.2 cm/s	
Robot diameter	10 cm	
Wheel radius	1.6 cm	
Detection range	5cm	Distance to the target for detection
Attack range	50 cm	Distance to the target for attacking
Perception_range	30 cm	perception range of the robots (for walls or neighbors)
Attack_strength	0.3	Reduction of target energy for robot attacking the target
Target_energy	100	Experiment ends when target energy is down to 0
Max time	900s	Max time of experiment

The global experience algorithm

Before implementing the two strategies for making the robots collaborate, it is necessary to set up a global algorithm that sequentially describes the course of the experiment. In the following, we will also describe the algorithms of the different methods used.

Overall, our experience can be described by the figure below

```
while target_energy>0 && total_time<900
   % UPdate total time
        total_time = total_time + r.time_step ;
    % Retrieve velocity and position of the robots
        x = r.get_poses();
        for i=1:N
            allRobots{i}.x = x(1,i);
            allRobots{i}.y = x(2,i);
            allRobots{i}.orientation = x(3,i);
        end
    % Distance to the target
        d_target = sqrt((x_target - x(1,:)).^2 + (y_target - x(2,:)).^2);
    % Walls detection
     %% Behavioural algorithm for the robots
        % Compute distance between robots
        dm = squareform(pdist([x(1,:)' x(2,:)']));
        for i = 1:N
            % Detect the neighbors within the perception range
                %neighbors = delta_disk_neighbors(x, i, PERCEPTION_RANGE);
                neighbors = find(dm(i,:)<=PERCEPTION_RANGE & dm(i,:)>0);
            % Collect local information for robot i
                % Walls information
                % Neighbors information (when applicable)
                % Target information (when applicable)
                    if (d_target(i)<DETECTION_RANGE)
                        allRobots{i}.set_info_cible(x_target, y_target);
                    end
                    if (allRobots{i}.cible_detected==1) && (d_target(i)<ATTACK_RANGE)
                        allRobots{i}.start_attack();
                    else
                        allRobots{i}.stop_attack();
                    end
              % Update the states of the robot
        end
        % Control LEDS
        % Update target energy
    % Display result
end
```

Fig. 5 The overall algorithm of the experiment

The decentralised strategy: random search

In this strategy, the robots move autonomously and individually in order to find and destroy the target, hence the name decentralised strategy. The figure below shows the swarm of twelve robots each trying without any group strategy to find the target (represented by the red dot).

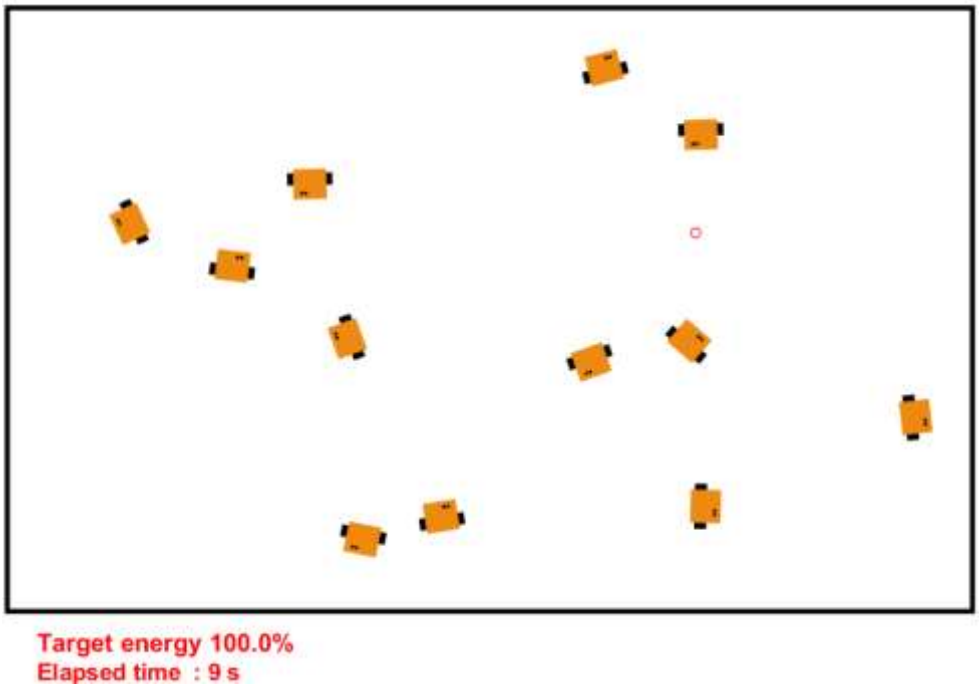

Fig. 6 Implementation of the decentralised random search strategy on Grishbots

However, each agent collaborates with the others by communicating its position or the position of the target if it finds it. This communication is possible if they are within the agent's perception radius.

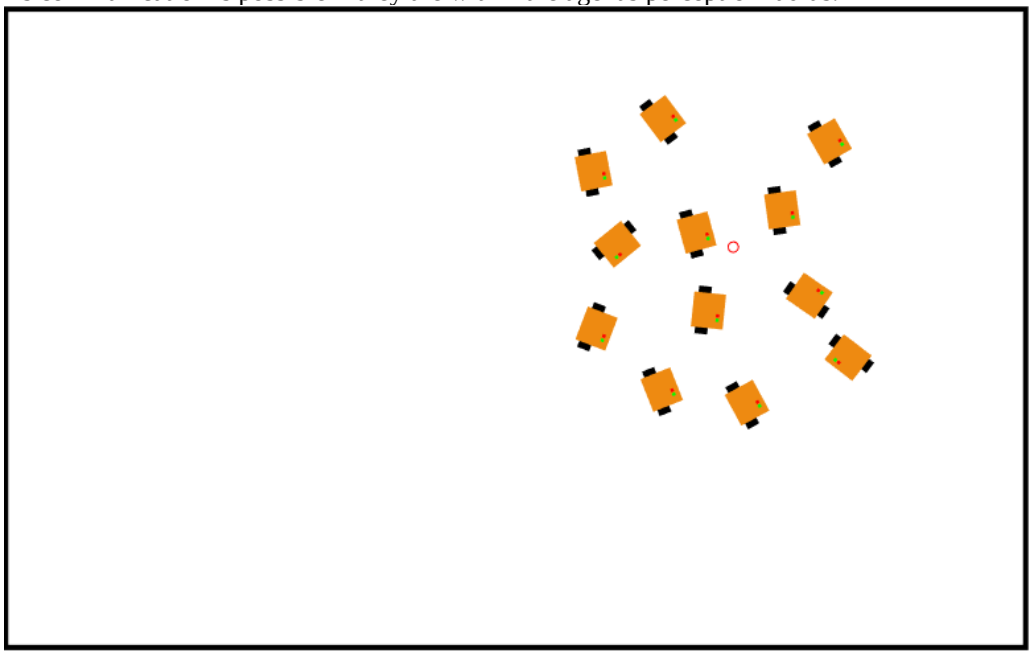

Fig. 7 The robots accumulate around the target to destroy it

After a time of 49s, the robots found and destroyed the target.

We can illustrate this strategy with the following algorithm

```
%  If the robot does not know where the target is,
% it searches by moving randomly
    if (robot.cible_detected==0)
        % First check whether there is a wall nearby.
        % if the robot is within 20 cm of a wall, it moves in the opposite direction
        border_v = [0 0];
        SAFE_BORDER = 0.2;
        % If the robot is stopped, it starts in a random direction
        if (robot.vx==0 && robot.vy==0)
            randVel = [rand.*2 - 1 ; rand.*2 - 1 ] ;
            v = randVel + border_v ;
            robot.move(v(1),v(2));
        else
            % Otherwise, it changes direction from time to time (at random)
                if rand<0.005
                    randVel = [rand.*2 - 1 ; rand.*2 - 1 ] ;
                    v = randVel + border_v ;
                    robot.move(v(1),v(2));
                else
                    robot.move(robot.vx + border_v(1) , robot.vy + border_v(2));
                end
        end
    else
    % The robot that knows the location of the target
     %gives the information to all its neighbours.
            for i=1:INFO.nbVoisins
                voisin = INFO.voisins{i};
                voisin.set_info_cible(robot.cible_x, robot.cible_y);
            end
    % If the robot knows the location of the target,
      it moves closer to it until it can attack it
        if (robot.cible_attacked==0)
            vx = robot.cible_x-robot.x ;
            vy = robot.cible_y-robot.y ;
            robot.move(vx,vy);
        else
            % If it is close enough to attack, it stops (automatic attack)
            robot.move(0,0);
        end
    end
```

Fig. 8 Algorithm illustrating the decentralised strategy

The centralised strategy: cluster attack

In this strategy, instead of moving the robots randomly and autonomously, we define a path that they will take in order to cover the arena as much as possible and try to find the target quickly. So the Grishbots will split into groups and cover the arena from one end to the other.

The figure below illustrates this behaviour

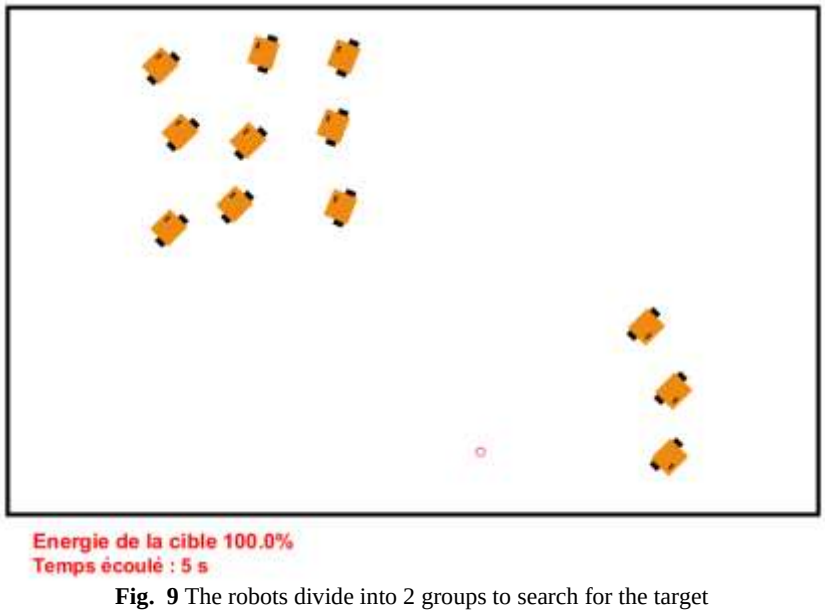

Fig. 9 The robots divide into 2 groups to search for the target

As illustrated in the figure above, each robot moves through the arena following the direction of its nearest neighbour. This allows our agents to stay together while maintaining a certain distance between them to avoid collisions.

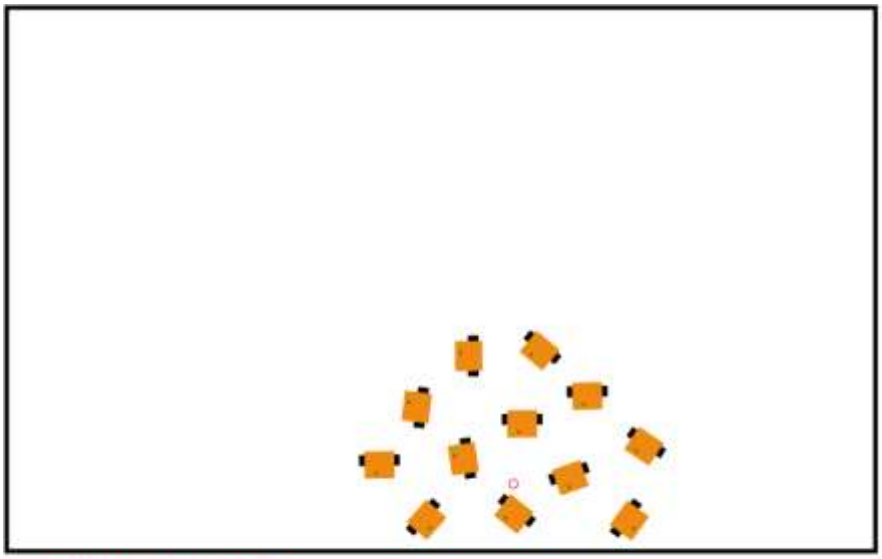

Fig. 10 The robots attack the target after having spotted it

The figure above shows the time taken by the robots to destroy the target, 136s.
We can illustrate this strategy with the following algorithm

```
% if the target has not been detected
if (robot.cible_detected==0)
    v = [0 ; 0]; %the robots start in the middle of the arena
    if (robot.vx==0 && robot.vy==0)
        %they move to an extremity
        if (robot.x>=0.0 && robot.y >= 0.0)
            v = [4 ; 4]; %
        end
        if (robot.x<=0.0 && robot.y <= 0.0)
            v = [-4 ; -4]; % bas à gauche
        end
        if (robot.x<=0.0 && robot.y>=0.0)
            v = [-4 ; 4]; % haut à gauche
        end
        if (robot.x>=0.0 && robot.y >= 0.0)
            v = [4 ; -4]; % bas à droite
        end
        robot.move(v(1),v(2));
    end
end
% Each robot follows the direction of its closest neighbour
courte = 0;
if (INFO.nbVoisins)
    for i=1:INFO.nbVoisins
        distx = (INFO.voisins{i}.x - robot.x)^2;
        disty = (INFO.voisins{i}.y - robot.y)^2;
        dist(i) = sqrt(distx + disty);
    end
    courte = 1;
    for i=1:INFO.nbVoisins
        if (dist(i) < dist(courte))
            courte = i;
        end
    end
     %if it encounters a wall,
     %it moves in the opposite direction
        border_v = [0 0];
        SAFE_BORDER = 0.2;
        if INFO.murs.dist_droite < SAFE_BORDER
            border_v = border_v + [-0.1 0];
        end
```

Fig. 11 Cluster attack algorithm

Comparison of the 2 methods

To compare the two methods, we carried out a series of forty measurements in order to take the time taken by the robots in each of them to destroy the target. The table below shows the results of these series

Table -2 Values in seconds of the time taken by the robots for the centralized method

centralized	88	80	86	62	33	77	56	57	40	45	61	36	42	40	40	40	99	53	69	49
	41	61	37	92	111	37	205	31	42	98	54	72	94	102	67	250	177	70	39	45

Table -3 Values in seconds of the time taken by the robots for the decentralised method

Decentralized	70	112	94	49	80	69	88	72	77	107	82	89	106	91	313	85	82
	94	77	96	54	84	71	97	110	130	75	166	109	159	135	215	197	108
	31	131	121	90	91	59											

The static parameters of these two series have been calculated and are summarised in the table below

Table -4 Statistical parameters of the data series

	Min	Max	Me	$\overline{x}$	σ	Q_1	Q_3
Centralized	31	250	61	72.7	45.56	41.75	86.5
Decentralized	31	313	91	104.25	49.46	77	110.5

Where $\overline{x}$ represents the Mean, σ the standard deviation, Q_1 the first quartile and Q_3 the third quartile.

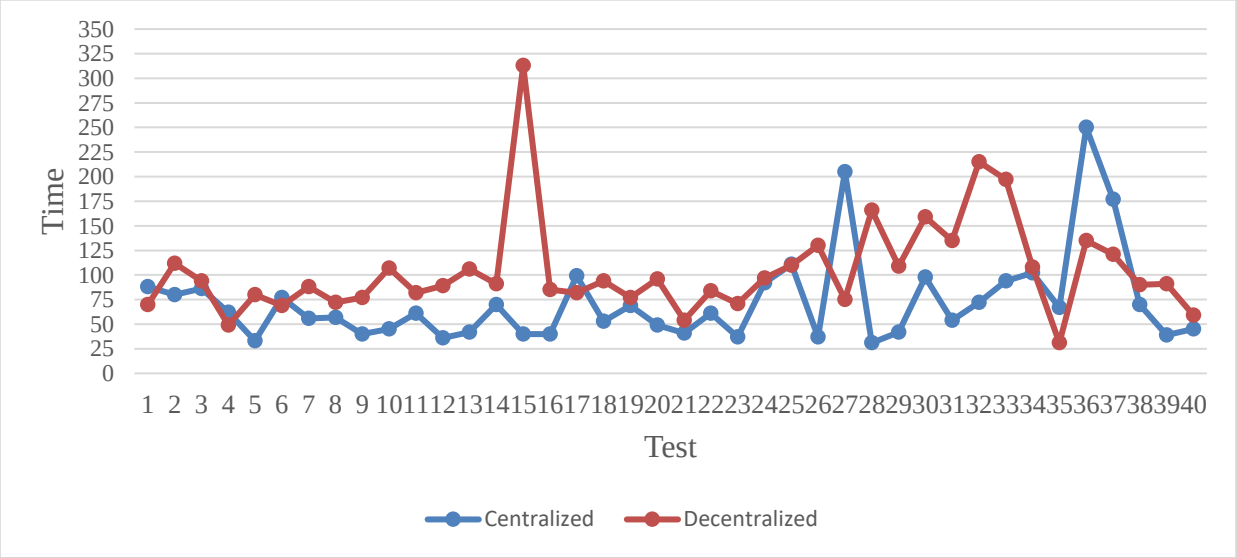

Fig. 12 Evolutionary curve of the times taken by the robots during the tests

The statistical data contained in Table 4 reveal that for the forty measurements carried out under the same conditions the minimum times of the two series are identical; however, the maximum time of the first is 250s while that of the second is 310s. Moreover, by observing the average values, we observe that in the centralised strategy, the robots take an average time of 72.7s to destroy the target, which is much less than the 104.25s taken in the decentralised strategy. This result is further supported by the comparison of the dispersion indicators Q_1 and Q_3 . We observe that 25% of the times taken are less than or equal to 41.75s and 75% are less than or equal to 86.5s for the first strategy. For the second strategy, 25% of the times taken are less than or equal to 77s and 75% are less than or equal to 110.5s. Finally, the comparison of standard deviations shows us that the times performed in the decentralised strategy are more dispersed or less homogeneous than those performed in the centralised strategy.

Based on this analysis, we can conclude that for an exploration problem the centralized method offers a better experience than the decentralized one. The agents browse the arena faster and take less time to find a target. This result is even more satisfying because the surface of the arena is constant and the number of robots is known, so it is easier for them to cover a larger surface if they are grouped.

REFERENCES

[1]. C. R. Kube and E. Bonabeau, "Cooperative transport by ants and robots," *Robotics and Autonomous Systems*, vol. 30, n° 1-2, pp. 85-101, Jan. 2000, doi: 10.1016/S0921-8890(99)00066-4.

[2]. J. Nembrini and A. Martinoli, "Robotics in Swarms, Results and Future Directions", p. 6.

[3]. G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, "Optimized flocking of autonomous drones in confined environments," *Sci. Robot.* , vol. 3, nº 20, p. eaat3536, Jul 2018, doi: 10.1126/scirobotics.aat3536.

[4]. D. Pickem *et al* , "The Robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1699-1706. doi: 10.1109/ICRA.2017.7989200.

[5]. D. Pickem *et al* , "Safe, Remote-Access Swarm Robotics Research on the Robotarium". arXiv, 3 April 2016. Accessed: 24 August 2022. [Online]. Available at: http://arxiv.org/abs/1604.00640

[6]. M. Rubenstein, A. Cornejo, and R. Nagpal, "Programmable self-assembly in a thousand-robot swarm," *Science*, vol. 345, nº 6198, pp. 795-799, August 2014, doi: 10.1126/science.1254295.

[7]. M. Dorigo, G. Theraulaz, and V. Trianni, "Reflections on the future of swarm robotics," *Sci. Robot.* , vol. 5, nº 49, p. eabe4385, Dec. 2020, doi: 10.1126/scirobotics.abe4385.

[8]. S. Mitri, S. Wischmann, D. Floreano, and L. Keller, "Using robots to understand social behaviour", *Biol Rev Camb Philos Soc*, vol. 88, nº 1, pp. 31-39, Feb 2013, doi: 10.1111/j.1469-185X.2012.00236.x.

[9]. K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, and G. C. H. E. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, nº 35, p. eaaw9710, Oct 2019, doi: 10.1126/scirobotics.aaw9710.

[10]. J. Krause, A. F. T. Winfield, and J.-L. Deneubourg, "Interactive robots in experimental biology," *Trends Ecol Evol*, vol. 26, nº 7, pp. 369-375, Jul 2011, doi: 10.1016/j.tree.2011.03.015.

[11]. L. Busoniu, R. Babuska, and B. De Schutter, "A Comprehensive Survey of Multiagent Reinforcement Learning", *IEEE Trans. Syst., Man, Cybern. C*, vol. 38, nº 2, pp. 156-172, March 2008, doi: 10.1109/TSMCC.2007.913919.

[12]. C. Yu, Y. Dong, Y. Li, and Y. Chen, "Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit," *J. eng.* , vol. 2020, nº 13, pp. 499-504, Jul. 2020, doi: 10.1049/joe.2019.1200.

[13]. A. T. Hayes and P. Dormiani-Tabatabaei, "Self-organized flocking with agent failure: Off-line optimization and demonstration with real robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, Washington, DC, USA, 2002, vol. 4, pp. 3900-3905. doi: 10.1109/ROBOT.2002.1014331.

[14]. R. Zlot and A. Stentz, "Market-based Multirobot Coordination for Complex Tasks," *The International Journal of Robotics Research*, vol. 25, nº 1, pp. 73-101, Jan. 2006, doi: 10.1177/0278364906061160.

[15]. R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", p. 551.

[16]. "Artificial Intelligence A-Z™: Learn How To Build An AI", *Udemy*. https://www.udemy.com/course/artificial-intelligence-az/ (accessed 18 September 2022).

[17]. M. van Otterlo, "Markov Decision Processes: Concepts and Algorithms", p. 23.

[18]. D. J. White, "A Survey of Applications of Markov Decision Processes", vol. 44, nº 11, p. 25.

[19]. H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning", p. 7.

[20]. A. Juliani, "Simple Reinforcement Learning with Tensorflow Part 0: Q-Learning with Tables and Neural Networks", *Emergent // Future*, 26 May 2017. https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0 (accessed 18 September 2022).