On the Choice of Utility Functions for Multi-Agent Area Survey by Unmanned Explorers

Lorenzo Pasini*, Achille Policante*, Daniele Rusmini*, Giulia Cisotto†, Elvina Gindullina*, and Leonardo Badia*
* Dept. of Information Eng. (DEI), University of Padova, Italy
† Dip. Informatica, Sistemistica e Comunicazione (DISCo), University Milano Bicocca, Italy
email: {lorenzo.pasini,achille.policante,daniele.rusmini}@studenti.unipd.it,
giulia.cisotto@unimib.it, {elvina.gindullina,leonardo.badia}@unipd.it

***Abstract*—The problem of multi-agent robotic survey of an unknown area is approached through a game theoretic framework. This is meant to enable cooperation in the group of robotic explorers reflecting their common objectives to minimize the effort in the surveying task, without requiring expensive exchanges of signaling. The game theoretic approach can be applied to avoid any preliminary planning, but just exploiting the ability of the robots to take smart actions based on the state of the environment and the behaviors of other neighboring agents. We discuss how the choice of different utility functions can improve the collaboration among the robots and lead to more efficient results.**

***Index Terms*—Game theory; Intelligent robots; Multi-robot systems; Simultaneous localization and mapping.**

I. Introduction

The exploration of environments is a relevant issue for unmanned ground and arial systems [1]–[4]. The increase in individual capabilities of single robots offers many opportunities for area surveying, which is a complex task if an unknown and unstructured environments are explored, and offers several applications for surveillance, path tracing, and general environmental monitoring [5]–[7].

If a single robotic agent is employed, these problems can be solved with adequate time and resources. But to speed up the process and make it more efficient, it is convenient to adopt multiple autonomous robots, which require coordination. The whole acquisition task becomes challenging when the individual efforts of the robots need harmonization. Game theory seems to be an efficient solution to this task, since it allows for distributed approaches, minimizing the requirements of information exchanges among the robots, and no explicit coordination [8]–[11]. Indeed, in a game theoretic setup, collaboration among multiple agents is just the byproduct of repeated interactions, all driven by selfish objectives in the individual agents, that are making autonomous decisions based on the available information [12].

We apply a game theoretic rationale to a multi-agent system where a number of non-adversarial, but not explicitly cooperative, robots, move in the same environment, being limited in mutual communications and available resources. We have been inspired by other projects [13], [14], whose methods to allow collaboration between drones for patrolling purposes are similar. On the other hand, there are some differences: robots in those tasks need to continuously patrol the interested area, while in our project the robots have to cover completely the map once in the fastest way possible.

Our problem can be reduced to exploring a randomly-generated grid of known size, where N robotic agents move and survey the area, avoiding obstacles. We model this task as a *dynamic game*, which is a standard setup, on which we apply the following idea. We can derive a full-fledged tree representation of the game, which considers every existing pattern. This is clearly unfeasible in an on-line setup, but we can preliminary perform it offline based on some general properties (such as number of obstacles, size of the map and so on), and apply the well-known game theoretic principle of *backward induction* to solve the game [15]. This allows considerable reductions in the search space, which are subsequently implemented in a local setup without full information [16], [17], but just with local awareness of the surrounding environment and the reciprocal positions of the other robots.

The algorithm is then implemented in a practical setup by considering individual agents as being selfishly driven by a utility function that nevertheless corresponds to a globally useful objective, i.e., surveying the area in an efficient and rapid manner, without any preliminarily agreed coordination. Then, we explore possible definitions of the utilities of the robots that make the implementation of our algorithm based on backward induction to be efficient [18]. In particular, we start from a naive representation of the utility being just the covered area, and we finetune it by adding more parameters such as keeping the robots separate and avoiding unexplored areas to be left behind. Eventually, these approaches are simulated multiple times by letting the games being played by autonomous agents [19], [20].

The results suggest that this procedure can lead to significant improvements at the price of an overall compact and simple implementation. However, different objectives need to be combined and the choice of the utility function is key. In particular, a contrast arises between a fast sweeping of the area and the goal of keeping the robots apart. This surely deserves further studies, nevertheless our analysis can be considered as a step in the proper direction to achieve an efficient solution by multiple distributed agents.

The rest of this paper is organized as follows. In Section II, we review the theoretical background of dynamic games of complete perfect information. Section III describes the specific game theoretic proposal of the paper, and Section IV presents its algorithmic implementation to solve the dynamic game at each time step. Section V discusses in more detail the choices of the utility functions and the numerical results. In Section VI, we draw the conclusions and elaborate on further possible developments.

II. Background

A. Dynamic games of complete perfect information

Dynamic games of complete perfect information are the simplest scenario of game theory where the interaction unfolds over multiple time instants. In this type of games, players are fully aware of all the past actions of the players, i.e., each information set in the game is a singleton [21].

In general, dynamic games can be represented in extensive form by defining the set of players, the utility function of each player, the order in which the players move, the actions allowed to the players every time they move, the information that they have when they move, and the probability of external events. All of the above must be common knowledge between all the players. The absence of external events and the common knowledge of all this information is the key characteristic of dynamic games of *complete* information. The extensive form of a game can be represented as a tree in graphical form, and for the case of complete information, this is a standard representation. We use the nodes of the tree to represent not only the stage of the game and the sequence of actions but also the information available to each player when moving. If each information set in the game is a singleton we call that game a dynamic game of complete *perfect* information.

B. Strategy profiles

In dynamic games of complete perfect information played by N players, a strategy profile for player $i \in \{1, 2, \ldots, N\}$ is a sequence of actions $s_i = \{a_1^i, ..., a_k^i\}$, which specifies the action of the player i for each information set of the player; since we are in a perfect information case, it means that an action is foreseen for each node of the tree. Then, we can define a global strategy profile as a set of the strategies of all players $\mathrm{s} = \{s_1, ..., s_N\}$.

C. Nash Equilibrium

The best response s_i^* of the player i to the vector of strategies of the other players $s_{-i} = \{s_1, ..., s_{i-1}, s_{i+1}, ..., s_N\}$ is the strategy of i that maximizes its utility given that s_{-i} is chosen by the others. Thus, s_i^* is s.t.

$$u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i}) \quad \forall s_i \in S_i \tag{1}$$

where S_i is the set of all possible strategies of player i. Given this formalism, we can define a *Nash equilibrium* (NE) as a global strategy profile, in which each player is choosing a best response. In other words, at an NE, given the strategies of all the other players, each player has no regrets about his choice [21]. NEs might lead to bad outcomes for all the players, which is known as the *tragedy of commons* [22], but in our case there is no real competition among the players, so the choice of an NE seems to be appropriate.

0	0	0	0	-100
0	-100	0	0	0
-100	-100	0	0	0
0	0	0	0	0
-100	0	0	0	-100

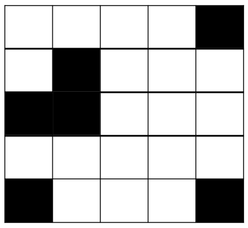

Fig. 1. Example of a small initialized map with randomly initialized obstacles. The map used in the problem is 20×20.

III. Problem Statement

A. The Map

We consider $N = 5$ robotic surveyors, which move in an area represented as a 20×20 bidimensional matrix, so that the robot positions are written as indices in the resulting grid. Obstacles in the map are represented by negative values in a particular matrix position. Positions that have already been explored by some robots are also tracked in the corresponding matrix entry. We have used both random initialization of obstacles positions and by-hand initialization to check the behavior of the robots in various conditions [9]. An example of a randomly initialized map can be seen in Fig. 1.

B. The Robots

First of all, we need to specify the information available to each robot at each time step. Each robot knows its absolute position, as well as the absolute position of all the other robots, and so the relative distance from them; also, the outer limit of the map is known, so robots are aware of when a move would lead to exiting the map. However, robots know the characteristics of the area only related to the closest cells to their current positions, i.e., the ones in which they can decide to go in the next time step. In this way, it can be known if these close squares are an obstacle or have already been visited by another robot. Importantly, robots do not know the entire map, they do not have any a priori knowledge of obstacle placements, but they can detect obstacles when they are near (for example using a proximity sensor), to avoid them [17].

The actions available to each robot at each time step are to move one step in the four directions, i.e., up, down, left, or right. Diagonal moves are not allowed. Finally, to fully characterize the robots, a utility function must be specified, which they use to compute the reward obtained by choosing a particular action [18]. This last element, described in Section V, is the core ingredient to obtain a good behavior of the robots, and we go through multiple options to improve its performance.

C. The decision process

The decision process is modeled as dynamic choices over multiple time epochs. At each step, the robots make a decision on the best action based on their available information (i.e., their positions, and the surrounding environment only). Yet, our proposal is to analyze the problem from a game theoretic standpoint, and to this end we assume that the game is a dynamic game of complete information. Then, we solve the game by backward induction to find one suitable NE, which has the further characteristic of being subgame-perfect [21]. As a final step, we update the robot positions and the state of the map according to the outcome of the backward induction. This process is repeated until the robots achieve full coverage of the map or the number of time steps hits a limit.

IV. The Backward Induction Algorithm

To solve the dynamic game at each time step (namely an action for each robot), we build the following algorithm. As a first step, a matrix containing all the possible joint strategies $s \in \mathcal{S}$ is computed. For each joint strategy in the matrix, we move all robots except the one that has to evaluate its utility, calculate it and all the procedure is repeated for each robot. The obtained utility matrix includes in each column the final payoff of each robot, corresponding to a particular combination of all the actions chosen by the robots. Thus, because we need to simulate the move of all the other robots for each possible choice of the one that we are considering, this part of the process is the slowest.

Algorithm 1 BackwardAlgorithm

```
 1: Parameters: $N$ (number of robots)
 2:             matrix $A$ (all joint strategies)
 3: Initialize $U(r, s)$, for all $s \in A$, $r \in \mathcal{R}$
 4: for $s^j \in A$ do
 5:     for all robots r do
 6:         Initialize map
 7:         Move all robots except $r$ following $s^j_{-r}$
 8:         update map
 9:         $U(r, s) \leftarrow u_r(s^j_r, s^j_{-r})$
10: Initialize $A' = A$
11: for robot r from N to 1 do
12:     for $s_{-r} \in A'$ do
13:         $s^*_r = \arg\max_a$
14:         $u_r(a, s^j_{-r})$
15:         Append $col_a(A')$ to $A'$
```

As a second step, the algorithm proceeds by backward induction to find a subgame-perfect NE [15], which is done as follows: it selects the first N (five, in our setup) columns of the matrix and checks the values of the payoffs in their last row. These are the possible payoffs of the last robot depending on its chosen action, assuming that the other robots have already made their strategic decisions. Thus, we can find the best action of the last robot and set it, after which, we can proceed in the same way for all the subsequent groups of N columns in the matrix. In the end, we have obtained a smaller matrix that corresponds to the payoff matrix at the penultimate level of the tree with the action of the last robot (considered in the previous step) fixed. Now, we apply the same reasoning to this smaller matrix and repeat it until only one column of the matrix remains. This last column is the one that corresponds to the NE actions and payoffs.

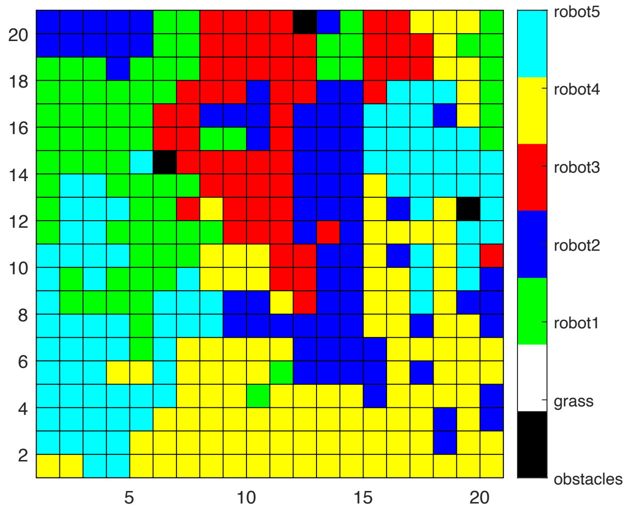

Fig. 2. Example of a final map obtained with the first implementation of the utility function

V. Choice of the Utilities and Numerical Results

How agents quantify the utility w.r.t. a particular action choice is a key element of the decision process in a game [18]. We start with a simple way to compute the utility for the robots and then we apply subsequent changes in the effort to reach full coverage of the area in a faster way.

A. First Implementation

In the first implementation, we evaluate the utility of a particular action only based on the value that is stored in the position that the robot would reach if choosing that particular action. So it is computed as follows.

$$u_i(a_j) = u_{hk} \tag{2}$$

where

u_i	= utility of player i
a_j	= jth action of player i
u_{hk}	= entry of the map matrix in row h, column k

In this way, if the action leads the robot against an obstacle, represented in the matrix with a negative value, that action yields an extremely low utility. Moreover, every time that a robot visits a particular position, the value stored in the corresponding map matrix entry is set to the standard negative value of -50. In this way, the robots can avoid visiting a position that has been already explored by someone else. If more than one action lead to the same utility value, the robots choose randomly between them.

To check the performance with this first implementation of the utility function, we run the simulation 100 times with 5 robots and a different randomly initialized 20×20 map each time. In this first case, we achieve a mean number of steps to obtain fully coverage of the area of ≈ 382 (most of the time the algorithm stop because it hits the maximum number of steps). In Fig. 2, we show an example of the final coverage of the map using this first type of utility function. In this figure, each square in the map is colored based on the last robot that has visited it. The black squares are obstacles.

However, the results are not satisfactory: even though the algorithm leads to full coverage of the area, there are multiple cases in which the algorithm stops because it hits the maximum number of epochs that are allowed. Moreover, it can be seen that we get a lot of isolated colored squares, which means that the robots are spending the majority of their time exploring positions in the map already explored by other robots.

B. *Second Implementation*

To solve the problems encountered in the first implementation, we introduce two additional elements in the computation of the utility corresponding to a particular action. The first one tries to include the distance from other robots as a relevant element in the utility calculation. Each robot computes the mean distance from other robots and increasing the value of this quantity is considered beneficial, so it improves the utility of the relative action. In this way, we are trying to stimulate our robots to select actions that increase the distance between them, still considering that visiting new positions is always better. The second change is to increase the penalty to visit a position that has been already visited many times. To do that we allow the robots to decrease the value stored in the map matrix corresponding to their actual position if that position has been already visited by someone else. So the utility is computed according to the following formula

$$u_i(a_j) = u_{hk} + u_i^d \tag{3}$$

where

u_i = utility of player i
a_j = j^{th} action of player i
u_{hk} = entry of the map matrix in row h, column k
u_i^d = utility for increasing the distance from other robots

We paid particular attention to how u_i^d is computed, and in the end we chose the following way

$$u_i^d = \frac{1}{N}\sum_{j \neq i}^{N} 10\log(||x - x_j|| + 1) - 50 \tag{4}$$

where

x_i = position of the ith robot (the one under consideration)
x_j = position of one of the other $N-1$ robots
N = number of robots

We used the logarithm in such a way that already distant robots have little influence in our action selection process. It is important to notice that positions are considered as the vectors containing the position indexes of the robots. After running the simulation for 100 more times, we obtain a mean number of steps to obtain full coverage of ≈ 172. Fig. 3 shows one of the final maps obtained with this second utility function implementation. This shows that this utility function can achieve a better division of the area and fewer number of steps to obtain full coverage.

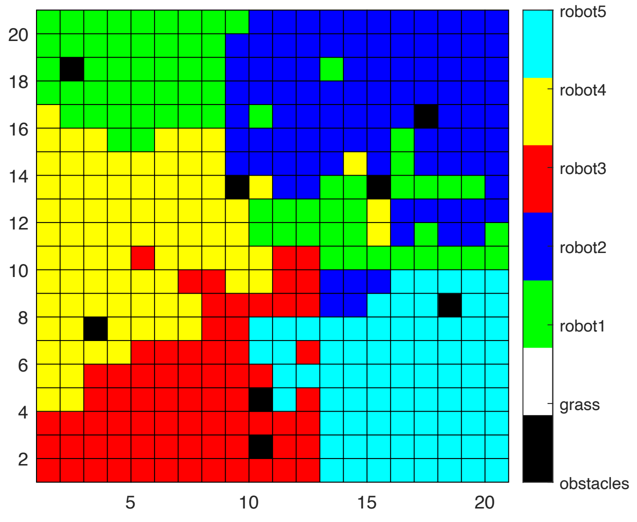

Fig. 3. Example of a final map obtained with the second implementation of the utility function

Fig. 4 shows the fractional coverage of the area with respect to the number of epochs (the number of simulation runs performed enables a very tight confidence). Evidently, a great portion of the time is used by the robots to search for the last squares that remain unexplored. The main problem is that the robots usually leave unexplored positions behind them, which are tedious to find at the end of the simulation. To try to solve this problem we introduce another change in the utility function, detailed in the next implementation.

C. *Third Implementation*

To further improve the performance, we introduce the last change in the computation of the utility function, whose aim is to increase the reward of visiting positions on the map that are next to already visited positions. By including this modification, we are trying to decrease the probability that a robot leaves an isolated unexplored position behind him.

To implement this, we proceed in the following way. Each time that a robot changes its position, it checks the eight nearest squares, and if they have not been visited yet, it increases a little bit the value of the corresponding map matrix entry. As usual, it also decreases the value of its actual position to mark it as an already visited position. It is important to underline that, even in this last implementation, all the modifications implemented in the second version of the utility function are kept. In Fig. 5, a typical map obtained using this third version of the utility function is shown. With this version of the utility function, we have obtained a mean number of steps ≈ 142 to obtain full coverage.

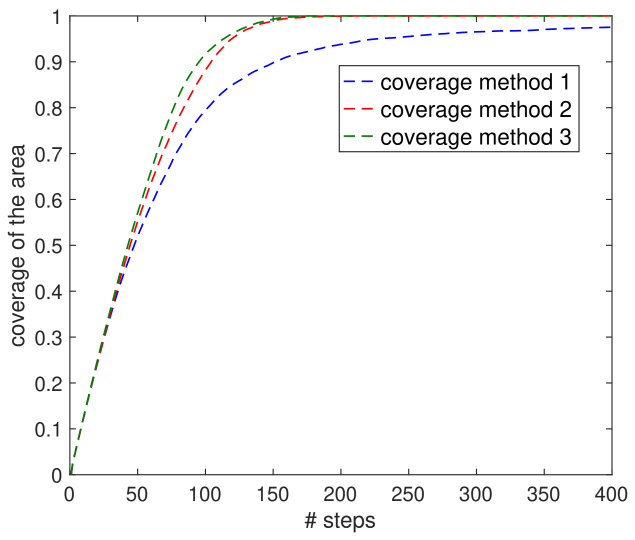

Fig. 4. Coverage versus time epochs, averaged over multiple runs, for a 20 × 20 grid. The confidence intervals are too small to be plotted

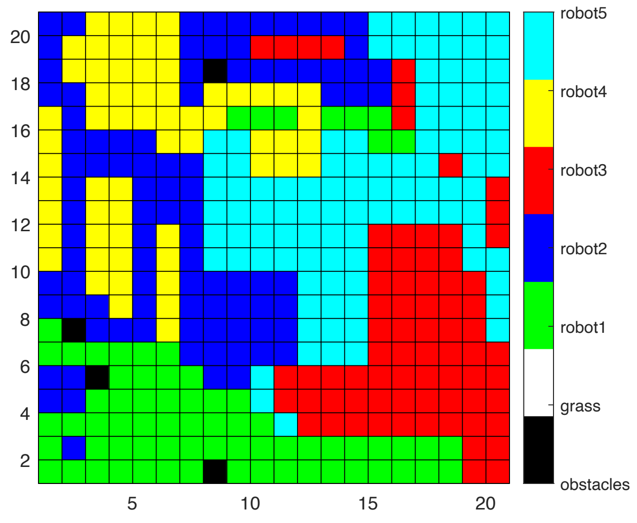

Fig. 5. Example of a final map obtained with the third implementation of the utility function

To compare the three approaches, check once again Fig. 4 for the fraction of covered area with respect to the number of epochs. The problem with Fig. 5 is that, despite the better performance, the map goes back to the principles of the first implementation contradicting the second one. So it is clear that this third implementation is not always able to improve the performance in the way that the second version of the utility function does. This is clearly visible in Fig. 6 where a comparison of the second and third utility functions is performed, and the former shows generally fewer repeated visits of a square.

In conclusion, both aspects must be carefully balanced, including the effort to push the robot to split the map into macro areas, but at the same time decreasing the number of times that one robot visits an already visited position, and also avoiding leaving areas behind.

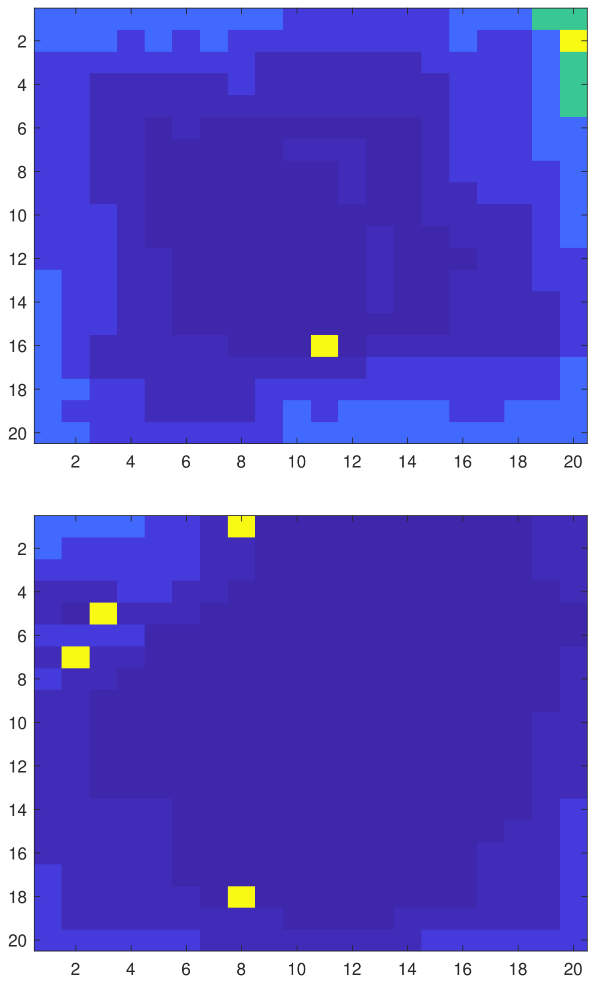

Fig. 6. Heatmap representation of the final map using the second (up) and the third (down) implementations of the utility function. Yellow squares are obstacles. Green and lighter blue squares represent highly visited positions.

VI. Conclusions and Future Work

We have explored the use of game theory to allow collaboration among robots that have a common goal of area surveying. The result obtained using the last implementation of the utility function could be considered good, still further changes could be explored to improve the final outcome.

A possible useful improvement would concern the robustness of the algorithm, especially avoiding loops and lengthy deadlocks. Even though our last implementation of the utility function was always able to avoid forced terminations due to reaching the maximum number of steps, the length of the path to fully cover a particular map of the same dimension can vary a lot from one experiment to another. So we ought to stabilize the number of steps that are needed to fully cover the map, and more in general, challenges of distributed implementations can be tackled [23], [24].

Other important changes may relate to the performance of the algorithm when the information available to the robots changes. For instance, this can be due to erroneous feedback in reporting the position between robots, or their communication range can be varied in such a way that the position of a far-away robot is unknown [16], [25]. Or, we can even address a possible variation in the field of vision of the robots to consider larger distances than nearby cells. All of these aspects can be included by allowing for a wider state space, which may require non-trivial modifications for scalability.

The last idea to improve the present work is to adjust the algorithm with the goal of increasing the size of the dynamic games, for example considering multiple moves for each robot. The main hurdle with this last modification is the computational effort needed to both build the tree and to solve it, but finding efficient ways to solve these two problems could be the primary goal of possible future work.

Acknowledgment

This work was supported by REPAC, a project funded by the University of Padova under the initiative SID-Networking 2019.

References

[1] A. K. Ray, L. Behera, and M. Jamshidi, "GPS and sonar based area mapping and navigation by mobile robots," in *Proc. IEEE Int. Conf. Ind. Inf.*, 2009, pp. 801–806.

[2] B. Pang, Y. Song, C. Zhang, and R. Yang, "Effect of random walk methods on searching efficiency in swarm robots for area exploration," *Applied Intelligence*, vol. 51, no. 7, pp. 5189–5199, 2021.

[3] M. Kazim, A. T. Azar, A. Koubaa, and A. Zaidi, "Disturbance-rejection-based optimized robust adaptive controllers for UAVs," *IEEE Syst. J.*, vol. 15, no. 2, pp. 3097–3108, 2021.

[4] S. Chien and K. L. Wagstaff, "Robotic space exploration agents," *Science Robotics*, vol. 2, no. 7, p. eaan4831, 2017.

[5] Y. Ai and C. Li, "Design of an indoor surveying and mapping robot based on SLAM technology," in *Proc. IEEE ICDSCA*, 2021, pp. 848–852.

[6] A. Marjovi, J. G. Nunes, L. Marques, and A. De Almeida, "Multi-robot exploration and fire searching," in *Proc. IEEE/RSJ Int. Conf. Intell. Rob. Sys.*, 2009, pp. 1929–1934.

[7] U. Baroudi, G. Sallam, M. Al-Shaboti, and M. Younis, "GPS-free robots deployment technique for rescue operation based on landmark's criticality," in *Proc. IEEE IWCMC*, 2015, pp. 367–372.

[8] G. A. Kaminka, D. Erusalimchik, and S. Kraus, "Adaptive multi-robot coordination: A game-theoretic perspective," in *Proc. IEEE ICRA*, 2010, pp. 328–334.

[9] E. Camuffo, L. Gorghetto, and L. Badia, "Moving drones for wireless coverage in a three-dimensional grid analyzed via game theory," in *Proc. IEEE APCCAS*, 2021, pp. 41–44.

[10] E. Semsar-Kazerooni and K. Khorasani, "Multi-agent team cooperation: A game theory approach," *Automatica*, vol. 45, no. 10, pp. 2205–2213, 2009.

[11] D. Gu, "A game theory approach to target tracking in sensor networks," *IEEE Trans. Syst., Man, Cybern. B*, vol. 41, no. 1, pp. 2–13, 2010.

[12] L. Badia, M. Levorato, F. Librino, and M. Zorzi, "Cooperation techniques for wireless systems from a networking perspective," *IEEE Wireless Commun.*, vol. 17, no. 2, pp. 89–96, 2010.

[13] E. Hernandez, J. del Cerro, and A. Barrientos, "Game theory models for multi-robot patrolling of infrastructures," *Int. J. Adv. Rob. Sys.*, vol. 10, 2013.

[14] P. E. U. de Souza, C. P. C. Chanel, and S. Givigi, "A game theoretical formulation of a decentralized cooperative multi-agent surveillance mission," in *Proc. DMAP Workshop*, June 2016, pp. 1–9.

[15] M. S. Malvankar-Mehta and S. S. Mehta, "Optimal task allocation in multi-human multi-robot interaction," *Optimiz. Lett.*, vol. 9, no. 8, pp. 1787–1803, 2015.

[16] E. Gindullina, E. Peagno, G. Peron, and L. Badia, "A game theory model for multi robot cooperation in industry 4.0 scenarios," in *Proc. IEEE APCCAS*, 2021, pp. 237–240.

[17] A. Kanakia, B. Touri, and N. Correll, "Modeling multi-robot task allocation with limited information as global game," *Swarm Intelligence*, vol. 10, no. 2, pp. 147–160, 2016.

[18] L. Badia and M. Zorzi, "On utility-based radio resource management with and without service guarantees," in *Proc. ACM MSWiM*, 2004, pp. 244–251.

[19] E. Gindullina, S. Mortag, M. Dudin, and L. Badia, "Multi-agent navigation of a multi-storey parking garage via game theory," in *Proc. IEEE WoWMoM*, 2021, pp. 280–285.

[20] E. Hernández, A. Barrientos, and J. del Cerro, "Selective smooth fictitious play: An approach based on game theory for patrolling infrastructures with a multi-robot system," *Expert Systems with Applications*, vol. 41, no. 6, pp. 2897–2913, 2014.

[21] S. Tadelis, *Game theory: an introduction*. Princeton university press, 2013.

[22] L. Prospero, R. Costa, and L. Badia, "Resource sharing in the Internet of things and selfish behaviors of the agents," *IEEE Trans. Circuits Syst. II*, vol. 68, no. 12, pp. 3488–3492, 2021.

[23] E. Gindullina, L. Badia, and X. Vilajosana, "Energy modeling and adaptive sampling algorithms for energy-harvesting powered nodes with sampling rate limitations," *Trans. Emerg. Telecommun. Technol.*, vol. 31, no. 3, p. e3754, 2020.

[24] G. Foderaro, S. Ferrari, and T. A. Wettergren, "Distributed optimal control for multi-agent trajectory optimization," *Automatica*, vol. 50, no. 1, pp. 149–154, 2014.

[25] L. Badia, "On the effect of feedback errors in Markov models for SR ARQ packet delays," in *Proc. IEEE Globecom*, Dec. 2009.