*robotics*

*Article*

# Detection of Hidden Moving Targets by a Group of Mobile Agents with Deep Q-Learning

**Barouch Matzliach [1,2,*], Irad Ben-Gal [1,2] and Evgeny Kagan [3]**

1 Department Industrial Engineering, Tel Aviv University, Tel Aviv 6997801, Israel; bengal@tauex.tau.ac.il
2 Laboratory for Artificial Intelligence, Machine Learning, Business and Data Analytics, Tel Aviv University, Tel Aviv 6997801, Israel
3 Department Industrial Engineering, Ariel University, Ariel 4076414, Israel; evganyk@ariel.ac.il
* Correspondence: barouchm@mail.tau.ac.il

**Abstract:** In this paper, we propose a solution for the problem of searching for multiple targets by a group of mobile agents with sensing errors of the first and the second types. The agents' goal is to plan the search and follow its trajectories that lead to target detection in minimal time. Relying on real sensors' properties, we assume that the agents can detect the targets in various directions and distances; however, they are exposed to first- and second-type statistical errors. Furthermore, we assume that the agents in the group have errorless communication with each other. No central station or coordinating agent is assumed to control the search. Thus, the search follows a fully distributed decision-making process, in which each agent plans its path independently based on the information about the targets, which is collected independently or received from the other agents. The suggested solution includes two algorithms: the Distributed Expected Information Gain (DEIG) algorithm, which implements dynamic Voronoi partitioning of the search space and plans the paths by maximizing the expected one-step look-ahead information per region, and the Collective Q-max (CQM) algorithm, which finds the shortest paths of the agents in the group by maximizing the cumulative information about the targets' locations using deep Q-learning techniques. The developed algorithms are compared against previously developed reactive and learning methods, such as the greedy centralized Expected Information Gain (EIG) method. It is demonstrated that these algorithms, specifically the Collective Q-max algorithm, considerably outperform existing solutions. In particular, the proposed algorithms improve the results by 20% to 100% under different scenarios of noisy environments and sensors' sensitivity.

**Keywords:** search and detection; path planning; decision making; mobile agents; group dynamics; neural networks; deep learning

check for updates

**Citation:** Matzliach, B.; Ben-Gal, I.; Kagan, E. Detection of Hidden Moving Targets by a Group of Mobile Agents with Deep Q-Learning. *Robotics* **2023**, *12*, 103. https://doi.org/10.3390/robotics12040103

Academic Editor: Charalampos Bechlioulis

Received: 25 June 2023
Revised: 9 July 2023
Accepted: 12 July 2023
Published: 14 July 2023

## 1. Introduction

Target searching is a fundamental problem in mathematics, which can be traced back to the origins of calculus [1] when it was considered a purely academic task. During World War II, the search problem for static and mobile targets became practical when Koopman [2] established a new research scheme to find German submarines in the Atlantic Ocean.

Formally, the target searching problem can be considered from two viewpoints. The first perspective requires distributing a given search effort over a search domain so that targets are detected with maximal probability. The second perspective requires planning search and navigation paths for the search agents such that they detect the targets with maximal probability in a minimal time.

In 1975, Stone presented the main concepts and ideas in his search theory [3] and substantiated optimizing techniques for distributing search efforts. In 1989, Washburn summarized the used search methods and presented formal techniques for tracking targets using mobile agents [4]. In later research works, search and screening methods, as well as target detection and tracking methods, were somewhat unified into a general probabilistic framework [5–8] that allowed the consideration of search and detection in different settings and with various levels of certainty.

Further developments in target-searching theory addressed multi-agent multi-target systems. However, along with the obvious advantages of the cooperative search, there is a considerable challenge to the high complexity of the involved algorithms for planning the optimal paths of a group of agents (the terms group, team, set and fleet are used interchangeably). To overcome such a challenge, cooperative search methods have implemented different heuristic approaches [7,9–11] and learning techniques [12–14].

In this paper, we consider the search for multiple static and moving targets by a group of mobile agents and propose new algorithms for path planning and agent navigation. Following conventional formulations, we consider a search with constrained paths [15] and simulate the detection uncertainty by using intermittently emitting targets [16]. Similar to previously obtained solutions [17,18], the agents utilize an occupancy grid [19,20], which represents their knowledge about the targets' locations, and we implement methods of deep Q-learning for planning the paths and navigating in the grid [21].

In contrast to other existing algorithms, the suggested solution considers the target-searching problem of mobile targets by a team of agents, such that detecting the targets includes statistical errors of the first and the second types. The solution includes two algorithms:

- A quick online reactive algorithm that implements dynamic Voronoi partitioning of the search domain and plans the search paths by maximizing expected one-step information gain in each region;
- A reinforcement learning algorithm that finds the shortest paths of the agents in the group by maximizing the cumulative information about the targets' locations using deep Q-learning techniques.

In the first algorithm, at each moment, the search domain is divided into Voronoi regions [22] with respect to the current probability map of the domain, and each agent searches in its region independently from the other agents. During the search, the location probabilities of the targets over the search domain change, causing the Voronoi regions to change, thus directly affecting the agents' search movements and plan over the updated regions. We call this version of the algorithm the Distributed Expected Information Gain (DEIG) algorithm in contrast to the previously developed versions of the centralized EIG algorithm [17,18] that did not use the Voronoi diagram. A clear benefit of this heuristic is its simplicity and low complexity, which allow online implementation over simple agents and components.

In the second algorithm, the decision regarding the next step of the agent is obtained via a deep Q-learning scheme over a neural network based on agent-by-agent value iteration [23]. The network receives the agent's location, the current probability map and the networks' parameters of previous agents as input and outputs the preferred move of the agent. This algorithm maximizes the network Q-value over a group of agents and is called the Collective Q-max algorithm.

The proposed algorithms are defined by a set of equations and are illustrated with numerical simulations that are compared with existing methods. These algorithms are implemented in the Python programming language using the PyTorch machine learning library. It is found that the novel deep Q-learning algorithm effectively governs the collective behavior of the search agents and substantially outperforms existing algorithms of collective detection without learning.

The rest of the paper is organized as follows. In Section 2, we introduce the required concepts and notation and formulate the problem. Section 3 is the main section where we present the suggested algorithms. Section 3.1 presents the algorithm of collective detection based on the Voronoi regions, and Section 3.2 presents the algorithm of collective detection using deep Q-learning. Section 4 describes the numerical simulations of the suggested algorithm and its comparisons with the known methods. Section 4.1 addresses the collective detection of static targets, and Section 4.2 addresses the detection of moving targets. In Section 4.3, we consider the training time required by the algorithm with deep Q-learning. Section 5 includes a general discussion about the suggested algorithms, and Section 6 concludes the paper.

## 2. Problem Formulation

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a finite set of cells that represent a grid over a two-dimensional domain. In the domain, there are $\xi$ targets, $\xi \leq n-1$, and $\eta$ agents, $\eta \leq n-1$ (in practice, $\eta \ll n$), each of which can be located in one cell. The agents are equipped with sensors that can detect close-enough targets, and accordingly, each agent plans its motion over the domain, aiming to detect the targets as fast as possible. The mobile targets, in contrast, are not aware of the agents and move independently of the agents' actions (i.e., we do not consider a game).

Each agent is equipped with sensors that can detect the targets in different directions and distances. Following a conventional detection sensor approach, as presented by Koopman [2,3], we assume that the detection probability of the target increases as the agent moves closer to the target and as the agent is exposed to the target location for a longer period of time. These assumptions are reflected by the simplified Koopman equation of the diction probability of a target in a cell:

$$Pr\{target\ detected\ in\ c_i \mid target\ located\ in\ c_i\} = 1 - exp[-\kappa(c_i, c_j, \tau)], \tag{1}$$

where $\kappa(c_i, c_j, \tau) \sim \tau / d(c_i, c_j)$ is the search effort applied to cell $c_i$ when the agent is in cell $c_j$; $\tau$ is the observation period; and $d(c_i, c_j)$ is the distance between cells $c_i$ and $c_j$. If all the cells are observed during the same period, $\tau$ can be omitted from the equation, as implemented below.

Moreover, we assume that the detection of a target is not perfect but is exposed to statistical errors of the first and second types, which implies that the agent can erroneously miss an existing target and can erroneously detect a target that does not exist in the cell.

To represent this assumption, the state of cell $c_i \in C$, $i = 1, 2, \ldots, n$, at time $t = 1, 2, \ldots$ is denoted by $s(c_i, t)$. Using occupancy grid techniques [19,20], the state $s(c_i, t)$ is considered to be a random variable with the values $s(c_i, t) \in \{0, 1\}$, such that $s(c_i, t) = 0$ indicates that cell $c_i$ at time $t$ is empty, and $s(c_i, t) = 1$ indicates that cell $c_i$ at time $t$ is occupied by a target. Because these two events are clearly complementary, their probabilities satisfy

$$Pr\{s(c_i, t) = 0\} + Pr\{s(c_i, t) = 1\} = 1. \tag{2}$$

We assume that the occupied cells at time $t$ broadcast an alarm signal $\widetilde{a}(c, t) = 1$ with the following probability:

$$p_{TA} = Pr\{\widetilde{a}(c, t) = 1 \mid s(c, t) = 1\}, \tag{3}$$

and the empty cells at time $t$ broadcast an alarm signal $\widetilde{a}(c, t) = 1$ with the following probability:

$$p_{FA} = Pr\{\widetilde{a}(c, t) = 1 \mid s(c, t) = 0\} = \alpha p_{TA}, \tag{4}$$

where $0 \leq \alpha < 1$. The first alarm is called a "true alarm", and the second alarm is called a "false alarm". The probabilities $p_{TA}$ and $p_{FA}$ are the probabilities of detection errors of the first and the second type, respectively.

Relying on Koopman Formula (1), the probability of perceiving the alarms is

$$Pr\{alarm\ percieved\ at\ c_j\ by\ agent\ k \mid alarm\ sent\ from\ c_i\} = \exp[-d(c_i, c_j)/\lambda^k], \tag{5}$$

where $\lambda^k$, $k = 1, 2, \ldots, \eta$, is the sensitivity of the sensor installed on the agent located in cell $c_j$.

The agents' knowledge about the targets' locations at time $t$ is represented by a probability vector $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$, where $p_i(t) = Pr\{s(c_i, t) = 1\}$ is the probability that, at time $t$, cell $c_i \in C$ of the domain is occupied by the target. The vector $P(t)$ is called the "probability map". We assume that all agents in the group are exposed to the same map $P(t)$ and share and update it in real time.

Accordingly, the probability of an event $\tilde{x}_j(c_i, t) = 1$, $i, j = 1, 2, \ldots, n$, implying that at time $t$, an agent $k$ located in cell $c_j$ receives a signal from cell $c_i$, is computed as follows:

$$Pr\{\tilde{x}_j^k(c_i, t) = 1\} = p_i(t-1)p_{TA}\exp[-d(c_i, c_j)/\lambda^k] + (1 - p_i(t-1))p_{FA}\exp[-d(c_i, c_j)/\lambda^k], \tag{6}$$

and the probability of the event $\tilde{x}_j(c_i, t) = 0$, i.e., implying that the agent does not receive a signal at time $t$ from that cell, is

$$Pr\{\tilde{x}_j(c_i, t) = 0\} = 1 - Pr\{\tilde{x}_j^k(c_i, t) = 1\}. \tag{7}$$

The event $\tilde{x}_j^k(c_i, t)$ represents a realistic assumption that the agent cannot distinguish between true and false alarms but only receives a signal, which can be either true or false.

Following the Bayesian scheme, when agent $k$ located in cell $c_j$ receives a signal from cell $c_i$, the probability that cell $c_i$ is occupied by the target is

$$Pr\{s(c_i, t) = 1 | \tilde{x}_j^k(c_i, t) = 1\} = \frac{p_i(t-1)p_{TA}}{p_i(t-1)p_{TA} + (1 - p_i(t-1))p_{FA}}, \tag{8}$$

and when this agent does not receive a signal from $c_i$, the probability that cell $c_i$ is occupied by the target is computed as follows:

$$Pr\{s(c_i, t) = 1 | \tilde{x}_j^k(c_i, t) = 0\} = \frac{p_i(t-1)(1 - p_{TA}exp[-d(c_i, c_j)/\lambda^k])}{p_i(t-1)(1 - p_{TA}exp[-d(c_i, c_j)/\lambda^k]) + (1 - p_i(t-1))(1 - \alpha p_{TA}exp[-d(c_i, c_j)/\lambda^k])}, \tag{9}$$

Therefore, the targets' location probabilities are:

$$p_i(t) = \begin{cases} Pr\{s(c_i, t) = 1 | \tilde{x}_j^k(c_i, t) = 1\} & \text{if agent } k \text{ received a signal at time } t, \\ Pr\{s(c_i, t) = 1 | \tilde{x}_j^k(c_i, t) = 0\} & \text{if agent } k \text{ had not received a signal at time } t. \end{cases} \tag{10}$$

Note that, if the true and false alarms are sent with equivalent probabilities $p_{TA} = p_{FA}$, then the agents' knowledge about the target location does not depend on the received alarms and is represented only by the probability map, as follows:

$$Pr\{s(c_i, t) = 1 | \tilde{x}_j^k(c_i, t) = 1\} = Pr\{s(c_i, t) = 1 | \tilde{x}_j^k(c_i, t) = 0\} = p_i(t-1). \tag{11}$$

The described process of receiving signals and updating the probability map is illustrated in Figure 1.

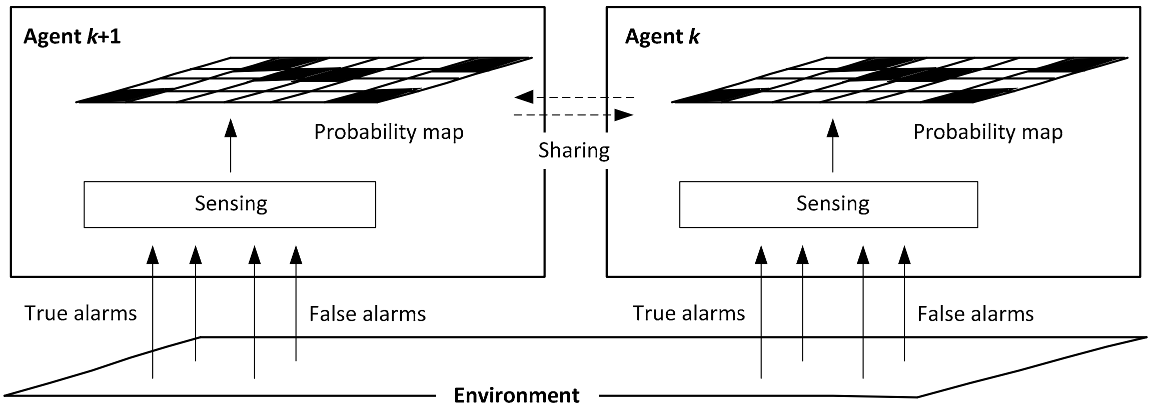


**Figure 1.** Receiving information and updating a shared probability map.

In the figure, each agent receives true and false alarms by its onboard sensors and updates the targets' location probabilities, which form a shared probability map.

In the case of static targets, the targets' location probabilities $p_i(t)$, $i = 1, 2, \ldots, n$, depend only on the agents' positions at time $t$ and their movements, and in the case of moving targets, these probabilities are defined both by the targets' and by the agents' motion.

Following the conventional formulation of search and detection problems [3,4], we assume that the targets are unaware of the agents' activities and move independently over the domain.

The agents' goal is to detect all $\xi$ targets in a minimal time. Note that, in the problem of detecting the targets, the agents are not required to chase the targets or to reach their locations physically but rather are required to specify the locations of the targets as definitively as possible by using the information obtained from their sensors.

## 3. Cooperative Detection: Using Voronoi Regions and Deep Q-Learning

The considered detection process follows the outline of a decision-making procedure and is specified as follows: at time $t$ in cell $c(t)$, each agent $k$ obtains the probability map $P(t)$, receives the alarms $\widetilde{a}(c, t)$ from the available cells and decides which cell $c(t+1)$ it should move to. Accordingly, a main challenge for an efficient cooperative search process is how to divide domain $C$ and distribute the search paths among $\eta$ agents and how to plan each agent's path to achieve a minimum detection time.

Below, we present two algorithms that address the challenge: the first considers the Voronoi regions of the agents and plans the agents' motion in their own regions, whereas the second uses the shared probability map and implements deep Q-learning techniques to control the agents' search.

### *3.1. Agents' Actions and Decisions*

Consider agent $k$ to be at time $t$ in cell $c(t) \in C$. We assume that the action $\mathbb{a}(t)$, which can be chosen by the agent, is one of nine possible movements from cell $c(t)$, which are "forward", "right-forward", "right", "right-backward", "backward", "left-backward", "left", "left-forward" and "stay in the current cell". In other words, the action of the $k$th agent is denoted by

$$\mathbb{a}^k(t) \in \mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}. \quad (12)$$

A probability map that represents the targets' locations at time $t+1$ is denoted by $P^k_{\mathbb{a}}(t+1)$, given that, at time $t$, the $k$th agent chooses action $\mathbb{a}^k(t)$. Then, given action $\mathbb{a}^k(t)$, the immediate expected informational reward of the $k$th agent is given by the Kullback–Leibler distance, namely

$$R^k_{\mathbb{a}}(t) = D_{KL}(P^k_{\mathbb{a}}(t+1)||P(t)), \tag{13}$$

between the expected agent's map $P^k_{\mathbb{a}}(t+1)$ and the current probability map $P(t)$. This informational reward $R^k(t)$ forms a basis for making decisions about the agents' next steps.

In the search by a single agent or by several agents making independent decisions over a shared probability map [18], the choice of the next action is governed by a simple rule:

$$\mathbb{a}^k(t) = \underset{\mathbb{a}\in\mathbb{A}}{\mathrm{argmax}} R^k_{\mathbb{a}}(t), \tag{14}$$

applied by each agent $k = 1, 2, \ldots, \eta$. This rule represents the agents' immediate one-step reaction to the changes in the probability maps and states of the targets.

In more complicated search cases by a single agent [21], the decision-making process considers the cumulative reward that is obtained in the sequence of the agent's actions. Given a policy $\pi$, which is a sequence of the agent's movements starting from its current position $c(t)$, the expected cumulative discounted reward obtained by the agent is

$$q_\pi(c(t), P(t), \mathbb{a}(t)) = \mathbb{E}_\pi\{\textstyle\sum_{\tau=0}^{\infty} \gamma^\tau R(\mathbb{a}, t+\tau)\}, \tag{15}$$

where the discount factor is $0 < \gamma \le 1$. The goal is then to find a maximum value

$$Q(c(t), P(t), \mathbb{a}(t)) = \underset{\pi}{max} q_\pi(c(t), P(t), \mathbb{a}(t)) \tag{16}$$

of the expected reward $q_\pi$ over the policies $\pi$ that can be obtained after action $\mathbb{a}(t)$ is chosen at time $t$.

Next, we extend reactive rule (13) and decision-making rules (15) and (16) to a search of multiple targets by a group of several interacting agents. The first approach is based on the Voronoi diagrams [22], and the second uses deep-learning techniques [21].

### *3.2. Reactive Decision Making in Voronoi Regions: A Distributed EIG Algorithm*

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a two-dimensional domain and $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$ be a probability map at time $t$. We assume that the map $P(t)$ is shared among all $\eta$ agents, $\eta \le n-1$, and is updated with respect to the information obtained by each $k$th agent, $k = 1, 2, \ldots, \eta$.

The Voronoi region for each agent $k$ is defined as follows. We assume that, at time $t$, the $k$th agent is in the cell $c^k(t)$, and $d(c^k(t), c)$ are the distances between cell $c^k(t)$ and cell $c \in C$ of the domain. Then, the Voronoi region of agent $k$ is subdomain $C^k(t) \subset C$ of domain $C$, where

$$C^k(t) = \{c \mid d(c^k(t), c) < d(c^j(t), c),\ j = 1, 2, \ldots \eta,\ j \ne k\}, \tag{17}$$

which includes the cells that are closer to the position of the $k$th agent than to the positions of the other agents. Because the agents change their positions with time, the Voronoi regions of the agents are updated accordingly.

The part of the probability map corresponding to the Voronoi region $C^k(t) \subset C$, $k = 1, 2, \ldots, \eta$, is denoted by $P^k(t) \subset P(t)$. The probability maps $P^k(t)$ are updated simultaneously with the updates of regions $C^k(t)$, and the values of the probabilities $p(t) \subset P^k(t)$ are updated according to the detection results. The set of Voronoi regions $C^k(t)$ is called the Voronoi diagram and is denoted by $\mathfrak{C}$, and the set of probability maps $P^k(t)$ is called the probability atlas and is denoted by $\mathfrak{P}$.

Based on the Voronoi regions, the detection process is conducted with several simple steps. Given positions $c^k(t)$ of agents $k = 1, 2, \ldots, \eta$, domain $C$ is divided into Voronoi regions $C^k(t)$. Each $k$th agent focuses on the corresponding part $P^k(t)$ of the probability map

$P(t)$, and rule (14) is used to make an independent decision regarding its next movement. When all $\eta$ agents make their decisions, it moves to the next positions $c^k(t+1)$ and observes the sensors' output. They update the location probabilities to $p(t+1)$ to form an updated probability map $P(t+1)$, and the process continues.

Formally, the Distributed EIG algorithm (Algorithm 1) based on the Voronoi regions is outlined as follows.

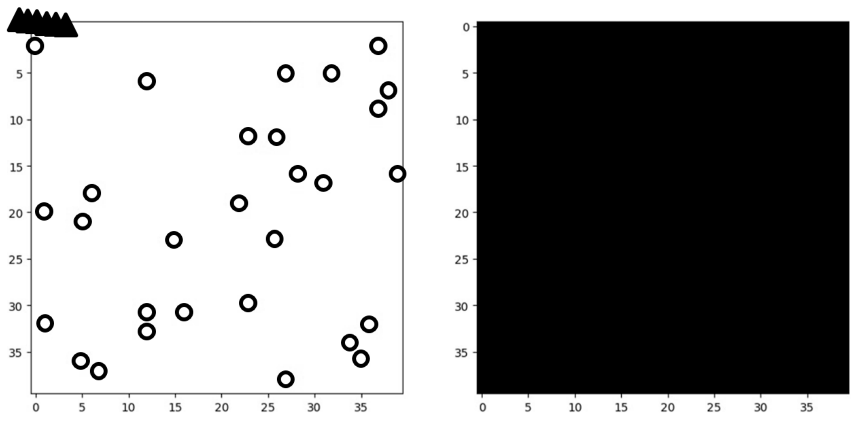
**Algorithm 1.** Cooperative detection with reactive decision making: Distributed EIG algorithm

```
Input: domain $C = \{c_1, c_2, \ldots, c_n\}$,
number of agents $\eta$,
initial agents' positions $c^1(0), c^2(0), \ldots, c^\eta(0)$,
set $\mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}$ of possible actions,
probability $p_{TA}$ of true alarms,
rate $\alpha$ of false alarms and their probability $p_{FA} = \alpha p_{TA}$,
sensor sensitivity $\lambda$,
initial probability map $P(0) = \{p_1(0), p_2(0), \ldots, p_n(0)\}$ on $C$,
number of targets $\xi$.
Output: target locations $\hat{c}^1(T),\ \hat{c}^2(T), \ldots, \hat{c}^\xi(T)$ at a termination time $T$.
1.  Start with $t = 0$, initial agent positions $c^1(t),\ c^2(t), \ldots, c^\eta(t)$ and initial probability map
    $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$.
2.  Create the Voronoi diagram $\mathfrak{C}(t) = \{C^1(t), C^2(t), \ldots, C^\eta(t)\}$, $C^k(t) \subset C$, $k = 1, 2, \ldots, \eta$.
3.  Create the probability atlas $\mathfrak{P}(t) = \{P^1(t), P^2(t), \ldots, P^\eta(t)\}$, $P^k(t) \subset P(t)$, $k = 1, 2, \ldots, \eta$.
Decision making
4.  For each agent $k = 1, 2, \ldots, \eta$, do:
5.  Choose action $\mathbb{a}^k(t) = \underset{\mathbb{a} \in \mathbb{A}}{\mathrm{argmax}} R^k_{\mathbb{a}}(t)$, where $R^k_{\mathbb{a}}(t) = D_{KL}(P^k_{\mathbb{a}}(t+1) || P^k(t))$.
6.  End for
Acting
7.  For each agent $k = 1, 2, \ldots, \eta$, do:
8.  Apply action $\mathbb{a}^k(t)$: move to new position $c^k(t+1)$.
9.  End for
Updating
10. Set $t = t + 1$.
11. For each agent $k = 1, 2, \ldots, \eta$, do:
12. Screen the domain $C$ with respect to the sensor's abilities.
13. Update the probability map $P(t)$ to $P(t+1)$.
14. End for
15. If all $\xi$ targets are detected, then
16. Set $T = t$ and terminate (go to line 20).
17. Else
18. Continue with line 2.
19. End if.
20. Return targets' locations $\hat{c}^1(T),\ \hat{c}^2(T), \ldots, \hat{c}^\xi(T)$
```

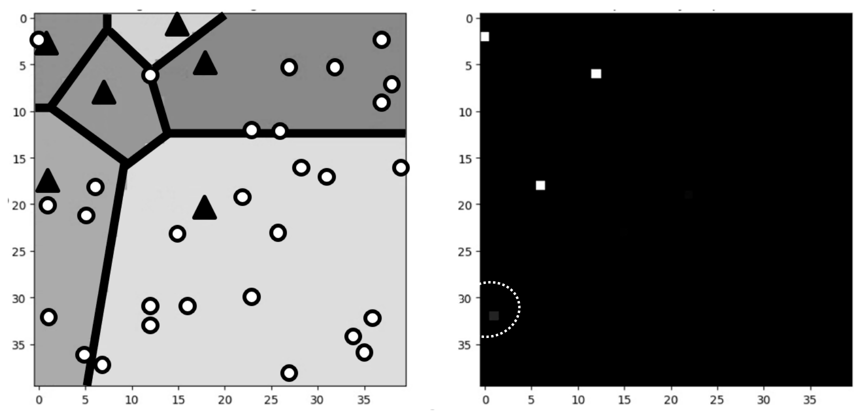
In the presented Algorithm 1, it is assumed that the number $\xi$ of targets is known, and this number is used to terminate the process. If the number of targets is unknown, then to define the termination, one can use certain measures over the probability map $P^*$, such as the entropy of the location probability, which represents sufficient knowledge about the targets' locations. In this case, the condition in line 15 is substituted or completed using the condition concerning the equivalence of the current probability map $P(t)$ and the objective map $P^*$. In the simulations below, we assume that $P^* = \left(p^*_1, p^*_2, \ldots, p^*_n\right)$ is constant with $p^*_i = 0.95$, $i = 1, 2, \ldots, n$, and we use this map as a termination condition. In the search with deep Q-learning, the objective probability map $P^*$ is used both for learning and termination; below, we consider this scenario in detail. Note that, in real-world search tasks, the map $P^*$

is not necessarily known to the agents, and they can continue their activity. However, it is necessary to control the performance of the algorithms.

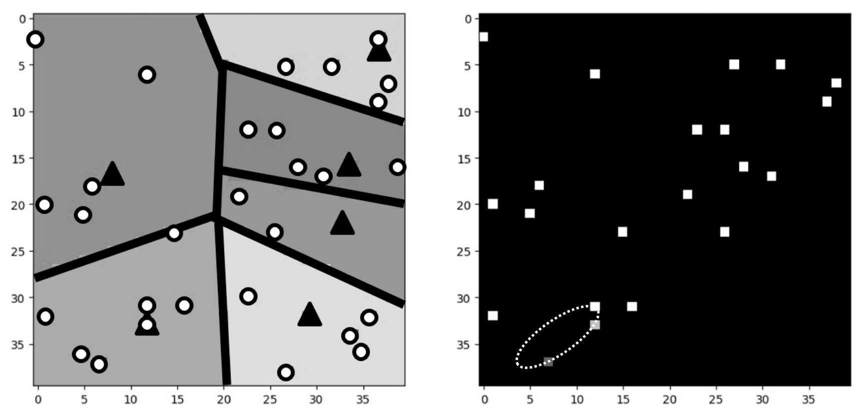
The activity of the Distributed EIG algorithm is illustrated in Figure 2, which shows four detection stages of $\zeta = 30$ static targets by $\eta = 6$ agents from a starting time $t = 0$ until $t = 90$, when all the targets are detected. The figures on the left side show the positions of the targets and the agents, and the Voronoi regions and the right-side figures show the probability maps. The detected targets are denoted by white squares, and the targets for which the detection probability is less than 0.95 are denoted by gray squares with the brightness proportional to the detection probability.



(**a**)

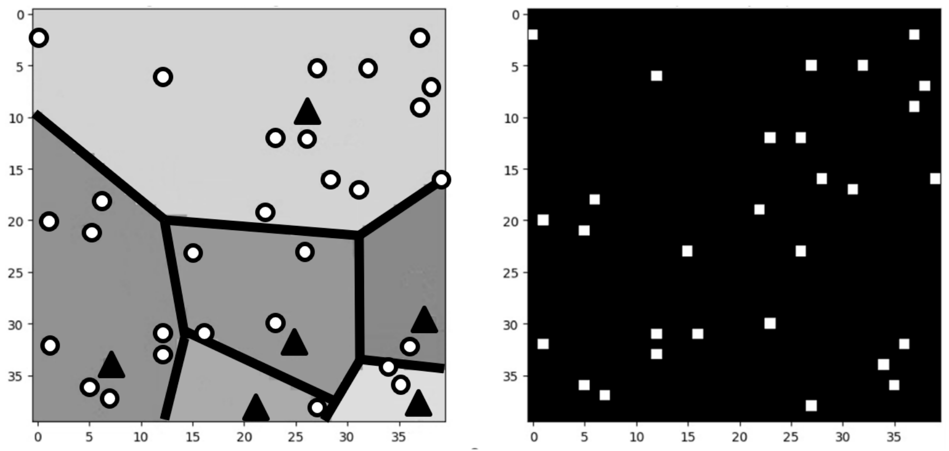


(**b**)



(**c**)

**Figure 2.** *Cont.*



(**d**)

**Figure 2.** Four stages of detection of $\xi = 30$ static targets (denoted by circles ○) by $\eta = 6$ agents (denoted by triangles ▲). The left-side figures show the positions of the targets and the agents with their Voronoi regions (denoted by gray areas), and the right-side figures show the probability maps, where white squares indicate the detected positions of the targets. (**a**) Agents start at the origin $(0, 0)$ and move over the domain. (**b**) At time $t = 30$, three targets are detected (the fourth target with low location probability is stressed by a dashed oval), and (**c**) at time $t = 60$, the agents detect 19 targets with probability greater than $p^* = 0.95$ (white squares) and 2 targets with lower probabilities (gray squares at points $(7, 37)$ and $(12, 33)$; stressed by a dashed oval). (**d**) At time $t = 90$, all targets are detected.

The presented Algorithm 1 is a direct extension of a previously developed algorithm [17,18] that maximizes the expected information gain over a domain $C$. However, as is demonstrated in Section 4, the use of a Voronoi diagram $\mathfrak{C}(t)$ and the independent activity of the agents in their Voronoi regions lead to changes in the agents' motion and a serious decrease in the detection time.

### 3.3. Collective Deep Q-Learning Approach

Now, we assume that, in addition to sharing the probability map, each agent makes its decisions with respect to the decisions made by the other agents. In this setting, a direct solution to the detection problem is computationally hard, and the exact activity of the group of agents can be specified only for small domains $C$. To overcome this problem, we apply agent-by-agent value iteration [23] together with reinforcement and deep learning techniques [24,25].

The suggested Collective Q-learning approach extends the Q-max search algorithm by a single agent described in detail [21]. It presents similar performances to those of the known optimal algorithms of search for moving targets based on dynamic programming techniques [26,27]. In particular, in comparison with the algorithms based on dynamic programming [26,27] and the algorithms with deep Q-learning, it was demonstrated that, in the search by a single agent and with a 10% false alarm ratio, the algorithm with deep Q-learning results achieves the same solutions as those of the known optimal dynamic programming algorithms. However, for 25% and 50% false alarm ratios, the algorithms with dynamic programming can plan only 7 steps for the agent in 2 h of simulation time, and the algorithm with Q-learning plans up to 32 steps. Consequently, the algorithm with Q-learning results in higher detection probabilities for the targets than those of the algorithm with dynamic programming: $p_1 = 0.99$ and $p_2 = 0.95$ versus $p_1 = 0.84$ and $p_2 = 0.68$ at best, respectively.

Then, based on the obtained results at a given time, the algorithm with Q-learning plans more agent steps and results in higher detection probabilities than those of the algorithm with dynamic programming. We extended the Q-learning algorithm for the search using multiple cooperating agents.

In contrast to the algorithms with dynamic programming [26,27] and the previously developed Q-max algorithm [21], the new Collective Q-max algorithm is not limited to searching for a single target in small domains; it defines the search by a group of agents acting in relatively large domains.

Each $k$th agent, $k = 1, 2, \ldots, \eta$, deals with two neural networks: the prediction network and the target or the Q-max network. The input layer of each of the networks includes $2n$ neurons, where $n$ is the size of the domain. The first chunk of $n$ inputs $(1, 2, \ldots n)$ receives a binary vector that represents the agent's position. If the agent is in cell $c_j^k$, then the $j$th input of the network is equal to 1, and the other $n - 1$ inputs are equal to 0. For convenience, the cell occupied by the considered agent is denoted by the value 10 instead of 1. The second chunk of $n$ inputs $(n + 1, n + 2, \ldots 2n)$ receives the target location probabilities; the $(n + i)$th input receives the target location probability $p_i$, $i = 1, 2, \ldots, n$, as it appears in probability map $P$.

The hidden layer of each network is a fully connected linear layer, which consists of $2n$ neurons and the sigmoid activation function $f(x) = 1/(1 + e^{-x})$ that returns values in the range $(0,\ 1)$. The other possibility is to use the SoftPlus techniques with the activation function $f(x) = \ln(1 + e^x)$ that returns values in the range $(0,\ \infty)$. In the simulations, it was observed that both functions result in similar performances. However, for both functions, the observed performance is significantly better than the performance based on the step activation function.

The output layer includes nine neurons with respect to the number $\#\mathbb{A}$ of possible actions: the first output corresponds to the action $\mathbb{a}_1 =$ "$\uparrow$", "step forward"; the second output corresponds to the action $\mathbb{a}_2 =$ "$\nearrow$", "step right-forward", up to action $\mathbb{a}_9 =$ "$\odot$", which implies "stay in the current cell". The value of the $i$th output is the maximal expected cumulative discounted reward $Q(c_j^k, P, \mathbb{a}_i^k)$ obtained by the agent if it is in cell $c_j^k$, $j = 1, 2, \ldots, n$, and given the probability map $P = (p_1, p_2, \ldots, p_n)$, it chooses action $\mathbb{a}_i^k$, $i = 1, 2, \ldots, 9$.

The scheme of the network of the $k$th agent is shown in Figure 3. For the input, the cells occupied by the agents have values of 1, and the cell occupied by the acting agent is denoted by 10.

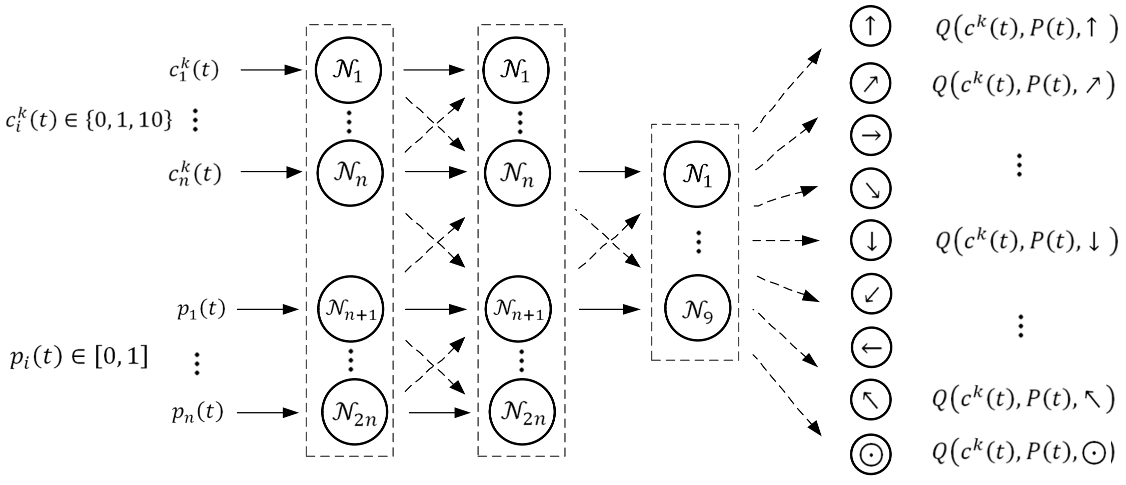


**Figure 3.** Scheme of the neural network used by the $k$th agent in the Q-max algorithm.

The interaction between the agents is conducted with two channels. The first uses the shared probability map $P$, and the second shares the updated weights of the links in the prediction network. This means that the $(k + 1)$th agent starts training the network with weights that were specified at the end of the training by the $k$th agent.

The Bellman equation used for calculating the maximal cumulative discounted reward is as follows:

$$Q\left(c^k(l), P(l), \mathbb{a}^k(l)\right) = \begin{cases} R^k_{\mathbb{a}}(l) + \gamma \underset{\mathbb{a}\in\mathbb{A}}{max} Q(c^{k+1}(l+1), P(l+1), \mathbb{a}) & k = 1, 2, \ldots, \eta - 1, \\ R^1_{\mathbb{a}}(l) + \gamma \underset{\mathbb{a}\in\mathbb{A}}{max} Q(c^1(l+1), P(l+1), \mathbb{a}) & k = \eta, \end{cases} \tag{18}$$

where $l = 1, 2, \ldots$ enumerates the steps at the learning stage. This equation forms a basis for updating the weights of the links in the networks. Thus, in the suggested Collective Q-max algorithm, the target network of the $k$th agent is considered a prediction network for the $(k+1)$th agent up to the $\eta$th agent, which is a predecessor of the $1^{st}$ agent.

The agents' rewards are calculated using the Voronoi diagram $\mathfrak{C}$. Each $k$th agent considers its region $C^k \in \mathfrak{C}$ and calculates the reward $R^k_{\mathbb{a}}$ within the relevant section $P^k$ of the probability map. Updating the Voronoi diagram $\mathfrak{C}$ is conducted after completing the calculations for all $\eta$ agents.

The SoftMax policy is implemented to select an action, where the probability $p(\mathbb{a}_i|Q;\theta)$ of choosing action $\mathbb{a}_i$ is defined as follows:

$$p(\mathbb{a}_i|Q;\theta) = \frac{\exp[Q(c, P, \mathbb{a}_i)/\theta]}{\sum_{j=1}^{9} \exp\left[Q\left(c, P, \mathbb{a}_j\right)/\theta\right]}, \tag{19}$$

where $\theta \in [0, +\infty)$ is a parameter that governs the randomness of the choice. If $\theta \to 0$, then $p(\mathbb{a}_i|Q;\theta) \to 1$ for $\mathbb{a}_i = \underset{\mathbb{a}\in\mathbb{A}}{\operatorname{argmax}} Q(c, P, \mathbb{a})$, and $p(\mathbb{a}_i|Q;\theta) \to 0$ for all other actions. If $\theta \to \infty$, then $p(\mathbb{a}_i|Q;\theta) \to \frac{1}{9}$, which corresponds to a randomly chosen action.

In the learning process, it is assumed that the value of the parameter $\theta$ decreases in the number of steps $l$ from its maximal value to zero. Then, in the first learning stages, the agent chooses actions randomly, and then, along with learning, the agent uses the information about the targets' locations learned by the networks. The first stages with randomly chosen actions are usually interpreted as exploration stages, and the later stages, based on the learned information, are considered exploitation stages.

Finally, $Q(c^k(l), P(l), \mathbb{a}^k(l); w)$ denotes the maximal cumulative discounted reward calculated at step $l = 1, 2, \ldots$ by the network with weights $w$, and $Q^+(c^{k+1}(l+1), P(l+1), \mathbb{a}^{k+1}(l+1); w')$ denotes the expected maximal cumulative discounted reward calculated using the vector $w'$ of the updated weights following the recurrent Equation (17). Note that the value of $Q^+$ is calculated for the next step $l+1$ and for agent $k+1$. Training of the networks is conducted using the temporal difference learning error:

$$\Delta(Q, l+k-1) = Q^+(c^{k+1}(l+1), P(l+1), \mathbb{a}^{k+1}(l+1); w') - Q(c^k(l), P(l), \mathbb{a}^k(l); w), \tag{20}$$

which allows sequential computation of the rewards obtained by the agents.

The learning process is illustrated in Figure 4.

The prediction network of the $k$th agent is used for choosing the action and specifying the expected position of this agent at step $l$, and the target network of the $k$th agent is used for calculating the reward after selecting and conducting the action that leads to step $l+1$. Then, the cumulative reward calculated by the target network is used to update the prediction network, and its weights are then used to update the weights of the trained network. Note that, together with the position of the $k$th agent, the target network considers the position of the $(k+1)$th agent. The target network of the $k$th agent, which at step $l$ is trained with respect to the positions of the $k$th and $(k+1)$th agents, is considered a prediction network of the $(k+1)$th agent at step $l+1$.

At the learning stage, we assume that the agents share the probability map, and that each agent updates the probabilities of the targets' locations over the entire domain. In addition, the reward of the $k$th agent is calculated over its Voronoi region. The Voronoi diagram is updated after updating the positions and the probability map by all $\eta$ agents.

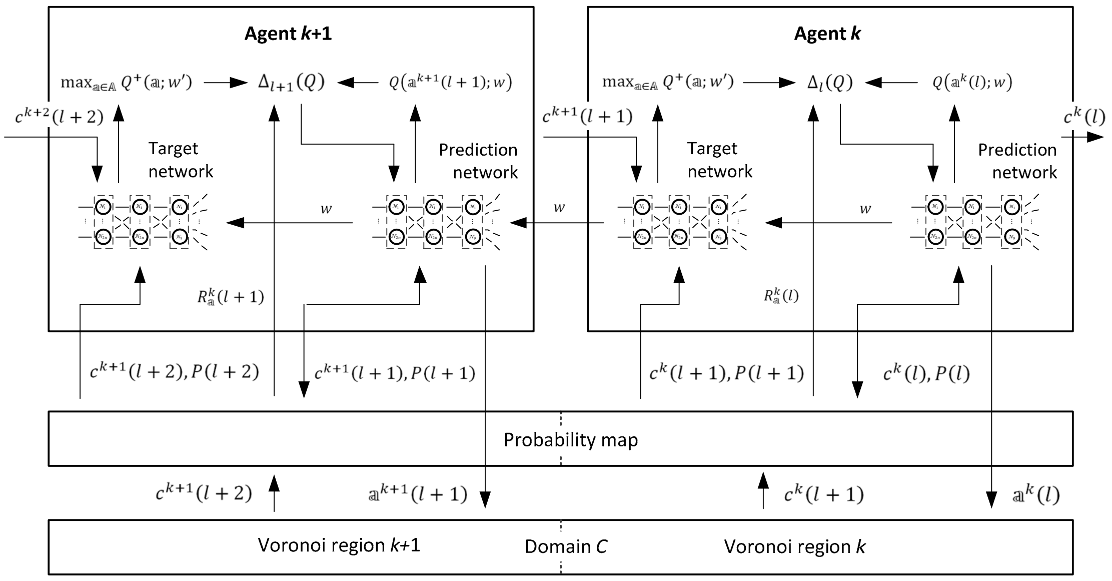


**Figure 4.** Actions of the offline model-based learning procedure of the Q-max algorithm.

In the above definitions, the cumulative rewards do not depend on the previous trajectories of the agents, and the process that governs the activity of each agent is a Markov process with states that include the positions of the agent and the probability maps. This property allows the use of the offline learning procedure. In this process, at step $l$, instead of the target location probabilities defined by Equations (8)–(10), the networks use the probabilities of the expected targets' locations $Pr\{s(c_i,l)=1|s(c_i,l-1)=1\}$ and $Pr\{s(c_i,l)=1|s(c_i,l-1)=0\}$ at step $l$ given the states of the cells in the previous step $l-1$. These probabilities are defined as follows using a Bayesian scheme:

$$Pr\{s(c_i,l)=1|s(c_i,l-1)=1\}= \\ =\frac{p_i(l-1)p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]}{p_i(l-1)(1-\alpha)+\alpha}+\frac{p_i(l-1)\left(1-\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)^2}{p_i(l-1)\left(1-p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)+(1-p_i(l-1))\left(1-\alpha p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)}, \quad (21)$$

$$Pr\{s(c_i,l)=1|s(c_i,l-1)=0\}= \\ =\frac{p_i(l-1)\alpha p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]}{p_i(l-1)(1-\alpha)+\alpha}+\frac{p_i(l-1)\left(1-\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)\left(1-\alpha p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)}{p_i(l-1)\left(1-p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)+(1-p_i(l-1))\left(1-\alpha p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda^k\right]\right)}, \quad (22)$$

Then, at the learning stage, instead of Equation (10), the targets' location probabilities are specified by the following condition:

$$p_i(l)=\begin{cases} Pr\{s(c_i,l)=1|s(c_i,l-1)=1\} & \text{if the target was in } c_i \text{ at } l-1, \\ Pr\{s(c_i,l)=1|s(c_i,l-1)=0\} & \text{if the target was not in } c_i \text{ at } l-1. \end{cases} \quad (23)$$

The learning process is terminated when the updated probability map $P(l)$ becomes equal to the objective map $P^*$. The detection process, similar to the considered above reactive procedures, can terminate either when the updated probability map $P(t)$ becomes equal to the objective map $P^*$ or when all $\xi$ targets are detected. Note that, in the learning scenario, the objective map $P^*$ is necessary for learning and for the control of the algorithm's performance.

The Collective Q-max algorithm used for the detection of multiple targets includes two stages: the learning stage, during which the agents' neural networks are trained, and the acting stage, which is an application of the agents (with the trained neural networks) for detecting the targets. The Algorithm 2 is outlined as follows.

**Algorithm 2.** Collective detection with deep learning: Collective Q-max algorithm

```
Network structure:
input layer: $2n$ neurons ($n$ agent positions and $n$ target location probabilities, both relative to the size $n$ of the domain),
hidden layer: $2n$ neurons,
output layer: 9 neurons (in accordance with the number of possible actions).
Activation function:
asymmetric sigmoid function $f(x) = 1/\left(1 + e^{-x}\right)$.
Loss function:
mean squared error (MSE) function.
Input: domain $C = \{c_1, c_2, \ldots, c_n\}$,
number of agents $\eta$,
sensor sensitivities $\lambda^1, \lambda^2, \ldots, \lambda^\eta$,
initial agents' positions $c^1(0), c^2(0), \ldots, c^\eta(0)$,
set $\mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}$ of possible actions,
probability $p_{TA}$ of true alarms,
rate $\alpha$ of false alarms and their probability $p_{FA} = \alpha p_{TA}$,
initial probability map $P(0) = \{p_1(0), p_2(0), \ldots, p_n(0)\}$ on $C$,
objective probability map $P^*$,
number of targets $\xi$ (or objective probability map $P^*$).
Output: target locations $\hat{c}^1(T),\ \hat{c}^2(T), \ldots, \hat{c}^\xi(T)$ at a termination time $T$.
Learning
1.  Generate training data set: agents' positions $c(0) = \left(c^1(0),\ c^2(0), \ldots, c^\eta(0)\right)$, probability
    map, $P(0) = \{p_1(0), p_2(0), \ldots, p_n(0)\}$.
2.  For each agent $k = 1, \ldots, \eta$, do:
3.  Create the prediction network.
4.  Create the target network as a copy of the prediction network.
5.  End for.
6.  Start with $l = 0$.
7.  For each pair $(c, P)$ from the training dataset, do:
8.  Create the Voronoi diagram $\mathfrak{C}(l) = \left\{C^1(l), C^2(l), \ldots, C^\eta(l)\right\}$.
9.  Create the probability atlas $\mathfrak{P}(l) = \left\{P^1(l), P^2(l), \ldots, P^\eta(l)\right\}$.
10. For each agent $k = 1, \ldots, \eta$, do:
11. For each action $\mathbb{a}^k \in \mathbb{A}$, do:
12. Calculate the value $Q(c^k(l), P(l), \mathbb{a}^k(l))$ with the prediction network.
13. Choose action $\mathbb{a}^k(l)$ with the value $p(\mathbb{a}|Q;\theta)$ and the SoftMax policy.
14. Apply the chosen action to the current position $c^k(l)$ and obtain the next position $c^k(l+1)$.
15. Update the probability map $P(l)$ to $P(l+1)$.
16. If $P(l+1) = P^*$, then
17. Set immediate reward $R^k_{\mathbb{a}}(l) = 0$.
18. Set cumulative reward $Q(c^k(l), P(l), \mathbb{a}^k(l), w) = 0$.
19. Else
20. Calculate immediate reward $R^k_{\mathbb{a}}(l)$ with respect to the probabilities of the agent's parts
    $P^k(l) \in \mathfrak{P}(l)$ and $P^k(l+1) \in \mathfrak{P}(l)$ of the probability map. {The Voronoi diagram and the
    cells associated with the $k$th agent remain, but the values of the probabilities change.}
21. End if.
22. End for.
23. Calculate the values $Q^+(c^k(l+1), P(l+1), \mathbb{a}^k; w')$ with the target network.
24. Calculate the temporal difference learning error $\Delta_l(Q)$ for maximal $Q^+$.
25. Update the prediction network with respect to the error $\Delta_l(Q)$.
26. Update the target network with the weights of the prediction network.
```

```
27. Update the prediction network of the $(k+1)$th agent with the target network of the $k$th agent.
28. End for.
29. If the training epochs ended
30. For each agent $k = 1, \ldots, \eta$, do:
31. Update the target network with the target network of the $\eta$t agent.
32. End for.
33. Start acting (go to line 36).
34. End if.
35. End for
Acting
36. Start with $t = 0$.
37. Obtain the initial agents' positions $c^1(t), c^2(t), \ldots, c^\eta(t)$.
38. Obtain the initial probability map $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$.
39. For each agent $k = 1, \ldots, \eta$, do:
40. Obtain the values $Q\left(c^k(t), P(t), \mathbb{a}^k(t), w\right)$ using the trained network.
41. Choose action $\mathbb{a}^k(t)$, which provides the maximum $Q\left(c^k(t), P(t), \mathbb{a}^k(t), w\right)$.
42. Apply the chosen action to the current position $c^k(t)$ and obtain the next position $c^k(t+1)$.
43. Screen the domain $C$. {The $k$th agent screens all of domain $C$ with respect to the abilities of the on-board sensors.}
44. Update the targets' locations $\hat{c}^1(t), \hat{c}^2(t), \ldots, \hat{c}^\xi(t)$.
45. Update the probability map $P(t)$ to $P(t+1)$.
46. End for.
47. If all $\xi$ targets are detected (or if $P(t) = P^*$), then
48. Set $T = t$ and terminate (go to line 53).
49. Else
50. Set $t = t + 1$.
51. Continue detection (go to line 39).
52. End if.
53. Return targets' locations $\hat{c}^1(T), \hat{c}^2(T), \ldots, \hat{c}^\xi(T)$.
```

The presented Algorithm 2 extends a previously developed detection algorithm by a single agent [21] and follows the same techniques for training the networks. However, instead of using individual prediction and target networks, it considers the prediction network of the next agent as its target network. Thus, each agent uses the training results of the previous agent and shares its knowledge with the next agent.

## 4. Numerical Simulations

The suggested algorithm 1 and algorithm 2 were implemented and tested in several settings, and their functionality was compared against a previously developed heuristic algorithm based on the same expected information gain.

Numerical simulations were implemented using the Python programming language, including the PyTorch machine learning library. The trials were executed on a PC Intel® Core™ i7-10700 CPU with 16 GB RAM (eight cores) with a GPU Nvidia GeForce GTX 1650Super (1280 CUDA Cores). Using this computer system, we measured the run time of the simulations over different datasets, demonstrating that the suggested algorithms can be implementable on conventional computers with CUDA parallel computing architecture and do not require specific parameters for their functionality.

In the Collective Q-max algorithm, the initial weights $w$ of neural networks were generated via the corresponding procedures of the PyTorch library. The optimizer used in the simulation was the ADAM optimizer from the PyTorch library. The size of the training data set was 100,000, the number of training epochs was 30, and the average time required for the offline training of the prediction network was approximately 10 h on the described computation platform. After an offline training period, online decision making was conducted directly by applying immediate selection without additional calculations.

In the simulations, we compared four algorithms: random search, in which the agents move randomly in the domain; centralized and Distributed EIG algorithms; and the Collective Q-max algorithm.

In all algorithms, we used $40 \times 40$ and $50 \times 50$ cell grid sizes, which correspond to practical military and civil tasks. In the case of military applications requiring search and detection in active operations, the size of the cell in the detection tasks is up to 0.5 km$^2$; thus, the grid of the indicated size represents a city district or the natural terrain of up to 1.0 km$^2$. In the case of civil applications, for example, in the search and rescue tasks and military logistic tasks, such grids represent the terrain of more than 6.0 km$^2$ while searching objects the size of a human and up to 25.0 km$^2$ while searching objects the size of an automobile.

The sensitivity of the sensor is specified by the parameter $\lambda$ (see Equation (5)), and in the simulations, we used the values $\lambda = 10$ and $\lambda = 15$, which are associated with the sensitivity of the Lidar sensor, for which the probability of detecting the target decreases from 1 (detection at a distance of 0 m) to $1/e$ (detection at a distance of 15 m). Certainly, in real-world tasks, the sensors' sensitivity $\lambda$ should be defined with respect to the type and quality of the sensor.

## *4.1. Detection of Static Targets*

The first set of simulations dealt with the detection of static targets. The results of these simulations are illustrated by the detection of $\xi = 30$ targets by $\eta = 6$ agents in the domain of size $n = 40 \times 40 = 1600$ cells. The probability of a true alarm is $p_{TA} = 1$, the sensor sensitivity is $\lambda = 15$, the initial target location probabilities are $p_i(0) = 0.05$ and $i = 1, 2, \ldots, n$, and the initial agent positions are $c^k(0) = (0, 0)$ and $k = 1, 2, \ldots, \eta$.

Let us consider the cumulated rewards obtained by the agents. The dependence of the cumulative reward on the time for the ratio of false alarms $\alpha = 0.5$ (see Equation (4)) is shown in Figure 5.

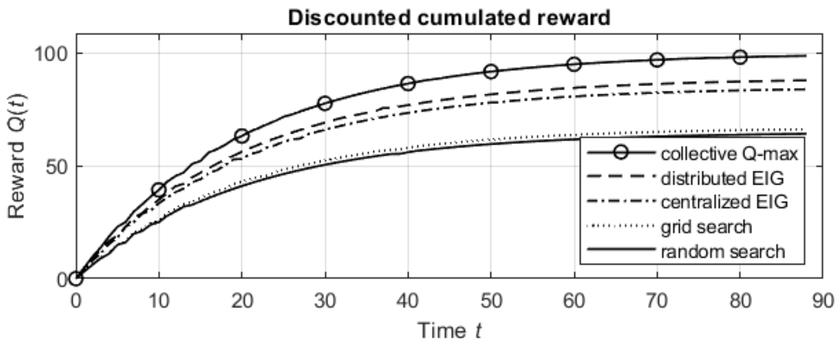


**Figure 5.** Dependence of the discounted cumulative reward on time; ratio of false alarms is $\alpha = 0.5$.

The fastest growth of the cumulative reward is provided by the Collective Q-max algorithm, and the slowest growth is provided by the random search method. The random search is slightly outperformed by the grid search, in which the paths of the agents are determined in advance. At the beginning of the search process at $t = 0$, the domain is divided into equal areas with respect to the number of agents, and each agent conducts the center of gravity search in its area according to the initial probability map. Finally, intermediate growth is provided by the EIG algorithms, for which the Distributed EIG algorithm outperforms the centralized EIG algorithm.

The same tendency was observed for the number of search actions (moves) conducted by the algorithms until detecting all the targets. The results of these simulations with different $\alpha$ ratios of false alarms, for which the sensitivity of the agents' sensors is defined by $\lambda = 15$, are summarized in Table 1.

**Table 1.** Number of actions up to the detection of 30 static targets by 6 agents for different ratios of false alarms. Sensor sensitivity is defined by $\lambda = 15$; the domain size is $40 \times 40$.

| **Detection Algorithm** | **Number of Actions up to the Detection of 30 Static Targets $n = 40 \times 40$, $\lambda = 15$** | | |
|---|---|---|---|
| | **$\alpha = 0.25$** | **$\alpha = 0.5$** | **$\alpha = 0.75$** |
| Random search | 120 | 175 | 450 |
| Grid search | 112 | 170 | 436 |
| Centralized EIG | 88 | 122 | 232 |
| Distributed EIG | 79 | 98 | 152 |
| Collective Q-max | 75 | 88 | 102 |

As expected, the worst results are obtained for a random search, in which the agents move randomly in the domain, whereas the best results are provided by the Collective Q-max algorithm. Note that the difference between the best results provided by the Collective Q-max algorithm and the other methods increases with the ratio of false alarms.

For comparison, Table 2 presents the number of search actions executed by the benchmarked algorithms to detect all the targets with a lower sensitivity of the agents' sensors given by $\lambda = 10$.

**Table 2.** Number of search actions up to the detection of 30 static targets by 6 agents for different ratios of false alarms. Sensor sensitivity is given by $\lambda = 10$; the domain size is $40 \times 40$.

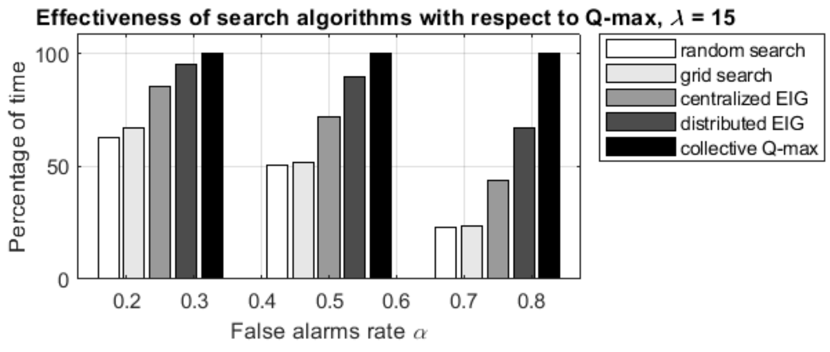
| **Detection Algorithm** | **Number of Actions up to the Detection of 30 Static Targets $n = 40 \times 40$, $\lambda = 10$** | | |
|---|---|---|---|
| | **$\alpha = 0.25$** | **$\alpha = 0.5$** | **$\alpha = 0.75$** |
| Random search | 205 | 310 | $> 600$ |
| Grid search | 198 | 305 | 590 |
| Centralized EIG | 145 | 221 | 321 |
| Distributed EIG | 134 | 210 | 285 |
| Collective Q-max | 106 | 138 | 178 |

Because the sensitivity $\lambda = 10$ of the agents' sensors is lower than that in the previous scenario, the agents need more search actions to detect the targets.

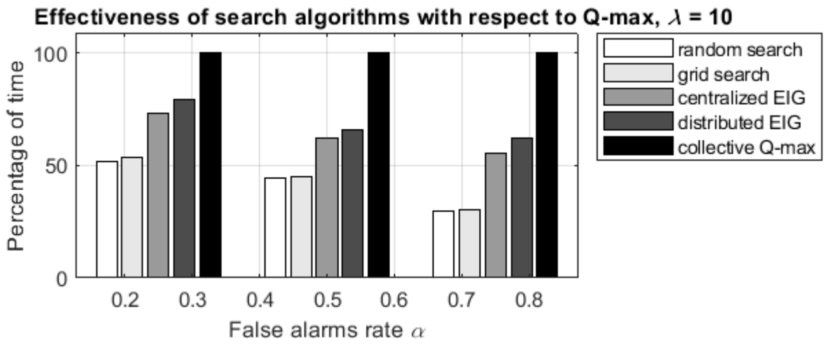
To stress the difference in the activity of the considered algorithms, Figure 6 illustrates the effectiveness of the search algorithms in comparison to Collective Q-max, which achieves a maximum efficiency of 100%.

The figure shows that the proposed Collective Q-max algorithm outperforms the other algorithms. The difference in the performance of the Collective Q-max algorithm and the other algorithms depends on the sensitivity of the sensors. For example, for $\lambda = 15$ and $\alpha = 0.25$, the Collective Q-max algorithm requires nearly two times less time to detect the targets than that of the random search, and for $\lambda = 15$ and $\alpha = 0.75$, it requires nearly three times less time than that of the random search. Similar ratios are observed for the other algorithms and values of $\lambda$ and $\alpha$.

The same relations were demonstrated in the other scenarios. For example, the number of actions conducted by the algorithms detecting up to 100 static targets by 10 agents equipped with sensors of sensitivity $\lambda = 10$ are presented in Table 3; the domain size is $n = 50 \times 50$.



(**a**)



(**b**)

**Figure 6.** Effectiveness of search algorithms in search of 30 static targets by 6 agents with two values of sensors' sensitivity $\lambda$: (**a**) $\lambda = 15$ and (**b**) $\lambda = 10$. The size of the domain is $n = 40 \times 40$.

**Table 3.** Number of search actions up to the detection of 100 static targets by 10 agents for different ratios of false alarms. Sensor sensitivity is given by $\lambda = 10$; the domain size is $50 \times 50$.

| **Detection Algorithm** | **Number of Actions up to the Detection of 100 Static Targets $n = 50 \times 50$, $\lambda = 10$** | | |
|---|---|---|---|
| | **$\alpha$ = 0.25** | **$\alpha$ = 0.5** | **$\alpha$ = 0.75** |
| Random search | 202 | 275 | $> 600$ |
| Grid search | 196 | 272 | $> 600$ |
| Centralized EIG | 156 | 215 | 322 |
| Distributed EIG | 135 | 198 | 305 |
| Collective Q-max | 102 | 135 | 165 |

Similar to the above, in this scenario, the Collective Q-max algorithm leads to essentially better results than those of the other algorithms, and the difference in the number of search steps required by the known methods and the suggested algorithm depends on the ratio of false alarms. It is seen that, for higher ratios of false alarms, this difference is higher. For example, for $\alpha = 0.75$, the difference in the search steps in the random search and in the suggested algorithm is greater than 400, and for $\alpha = 0.25$, this difference is 100. Therefore, when considering the detection of static targets, the suggested Collective Q-max algorithm is shown to be preferable with respect to the other methods, and such preference is more significant in tasks with a high ratio of false positive errors. If the ratio of false positive errors is relatively low, then the suggested Distributed EIG algorithm can also be applied.

### *4.2. Detection of Moving Targets*

The second set of simulations dealt with the detection of moving targets. As above, we present the results of simulations with $\xi = 30$ targets searched by $\eta = 6$ agents in a domain of size $n = 40 \times 40 = 1600$ cells. The probability of a true alarm is $p_{TA} = 1$, the sensor sensitivity is given by $\lambda = 15$, the initial targets' location probabilities are uniformly distributed with $p_i(0) = 0.05$, $i = 1, 2, \ldots, n$, and the initial agents' positions are $c^k(0) = (0, 0)$, $k = 1, 2, \ldots, \eta$. For simplicity, here, we consider slowly moving targets such that the probability of staying in the same cell is substantial, $Pr\{\hat{a}(t) = \odot\} = 0.9$, and the probability of taking a search step in any direction is $Pr\{\hat{a}(t) \in \mathbb{A}\backslash\odot\} = \frac{1-Pr\{\hat{a}(t)=\odot\}}{\#\mathbb{A}-1} = 0.0125$.

For convenience, we consider the results of the simulations with the same sensor sensitivities, i.e., with $\lambda = 15$ and $\lambda = 10$ and three ratios of false alarms. However, in this study, we limit the false alarm ratios to lower values: $\alpha = 0.1$, $\alpha = 0.15$ and $\alpha = 0.25$. The reason for this is that, in scenarios with higher false alarm ratios, the detection process takes too much time, and the resulting high numbers of search actions, despite the use of the same relations, are less illustrative.

The results of the simulations with different false alarm ratios $\alpha$ and a fixed sensitivity of the agents' sensors are given by $\lambda = 15$. These results are summarized in Table 4.

**Table 4.** Number of search actions up to the detection of 30 moving targets by 6 agents for different ratios of false alarms. Sensor sensitivity is given by $\lambda = 15$; the domain size is $40 \times 40$.

| **Detection Algorithm** | **Number of Actions up to the Detection of 30 Moving Targets $n = 40 \times 40$, $\lambda = 15$** | | |
|---|---|---|---|
| | **$\alpha = 0.1$** | **$\alpha = 0.15$** | **$\alpha = 0.25$** |
| Random search | 205 | 310 | 720 |
| Grid search | 190 | 301 | 710 |
| Centralized EIG | 140 | 208 | 366 |
| Distributed EIG | 132 | 180 | 340 |
| Collective Q-max | 115 | 132 | 188 |

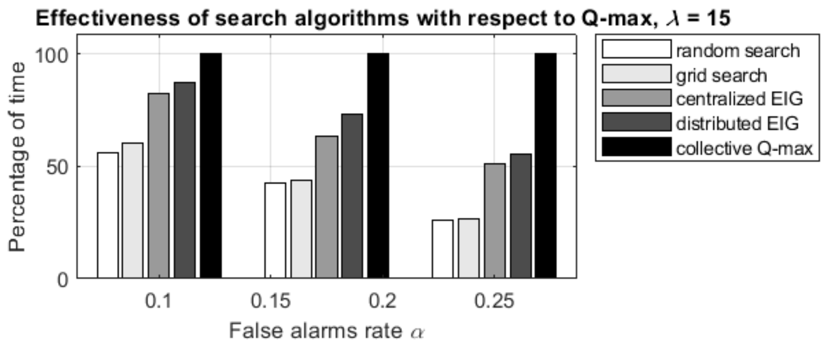
As seen above, the worst results correspond to the random search, and the best results are provided by the Collective Q-max algorithm. In addition, note that, for moving targets, when $\alpha = 0.25$, the random search method requires six times more actions (720) than those of the case of the detection of static targets (120, see Table 1). For the same $\alpha$ value, the EIG algorithm requires nearly four times more search actions (366 vs. 88 and 340 vs. 79). Finally, the ratio between these numbers of required actions between the static and the dynamic case when applying the suggested Collective Q-max algorithm is only $\frac{188}{75} \approx 2.5$.

For comparison, Table 5 presents the number of search actions conducted by the algorithms until detecting all targets, with lower sensitivity for the agents' sensors, $\lambda = 10$. As expected, because the sensitivity of the agents' sensors is given by $\lambda = 10$, the ratios between the number of required actions are lower compared to this ratio in the previous scenario, and the agents need more actions to detect the targets. In addition, the number of search actions follows the same tendency as above. Thus, the suggested Collective Q-max algorithm strongly outperforms all the other methods.

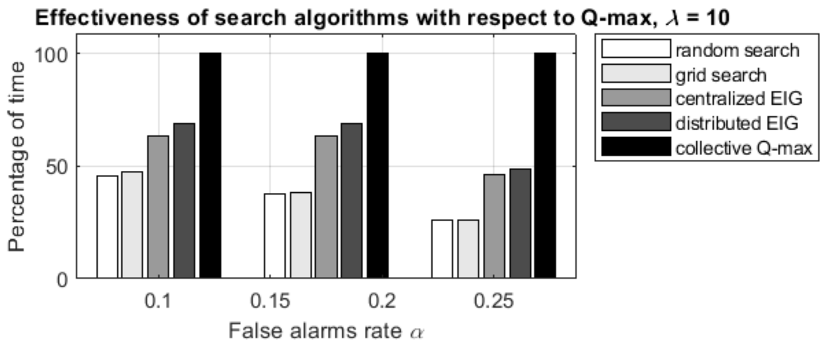
The difference in the activity of the considered algorithms is exemplified in Figure 7.

**Table 5.** Number of search actions required up to the detection of 30 moving targets by 6 agents for different probabilities of false alarms. Sensor sensitivity is given by $\lambda = 10$; the domain size is $40 \times 40$.

| **Detection Algorithm** | **Number of Actions up to the Detection of 30 Moving Targets $n = 40 \times 40$, $\lambda = 10$** | | |
|---|---|---|---|
| | **$\alpha = 0.1$** | **$\alpha = 0.15$** | **$\alpha = 0.25$** |
| Random search | 280 | 410 | 850 |
| Grid search | 268 | 405 | 844 |
| Centralized EIG | 202 | 245 | 475 |
| Distributed EIG | 185 | 225 | 450 |
| Collective Q-max | 128 | 155 | 220 |



(**a**)



(**b**)

**Figure 7.** Effectiveness of search algorithms in search of 30 moving targets by 6 agents with two values of sensors' sensitivity $\lambda$: (**a**) $\lambda = 15$ and (**b**) $\lambda = 10$. The domain size is $n = 40 \times 40$.

Finally, parallel to the results presented in Table 3, the number of actions executed by the algorithms up to the detection of 100 moving targets by 10 agents equipped with sensors of sensitivity $\lambda = 10$ is presented in Table 6. The size of the search domain is $n = 50 \times 50$. In this scenario, the Collective Q-max algorithm also obtains the best results in comparison with the other algorithms, and its advantage is higher as the ratio of false alarms increases.

**Table 6.** Number of search actions required up to the detection of 100 moving targets by 10 agents for different ratios of false alarms. Sensor sensitivity is given by $\lambda = 10$; the domain size is $50 \times 50$.

| **Detection Algorithm** | **Number of Actions up to the Detection of 100 Moving Targets $n = 50 \times 50$, $\lambda = 10$** | | |
|---|---|---|---|
| | **$\alpha = 0.1$** | **$\alpha = 0.15$** | **$\alpha = 0.25$** |
| Random search | 295 | 422 | 865 |
| Grid search | 285 | 411 | 861 |
| Centralized EIG | 205 | 247 | 465 |
| Distributed EIG | 190 | 232 | 447 |
| Collective Q-max | 132 | 162 | 225 |

## *4.3. Learning Errors and Run Time of the Collective Q-Max Algorithm*

We consider two characteristics of the suggested Collective Q-max algorithm, which emphasize the feasibility of its implementation in solving real-world problems.

As indicated above, in the presented simulations, we used 30 learning epochs with 100,000 training data sets. This choice of parameters is based on the dependence of the percentage of learning errors on the number of learning epochs. The graph of this dependence is shown in Figure 8.

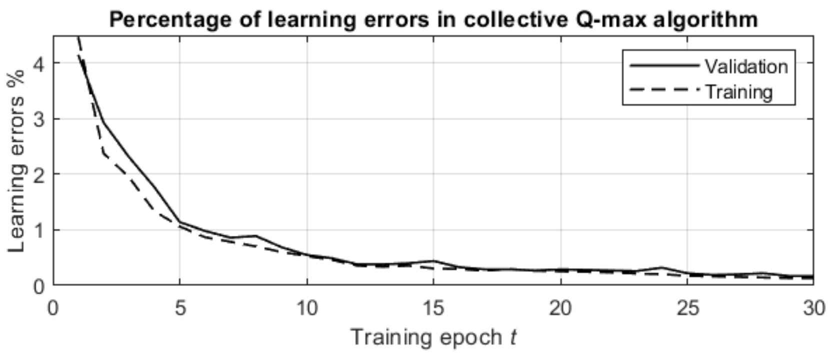


**Figure 8.** Dependence of the percentage of learning errors on the number of training epochs.

The percentage of learning errors decreases exponentially and converges to the value 0.1% after 30 training epochs.

Next, we consider the training run time. The run times required for training the neural networks in different simulation settings for the abovementioned computation system are summarized in Table 7.

**Table 7.** The training run times and learning errors for different data sets and search domains.

| **Domain Size $n_x \times n_y$** | **Number of Nonzero Weights in the Neural Network** | **Size of the Data Set** | **Run Time for One Epoch [minutes]** | **Mean Squared Error *** |
|---|---|---|---|---|
| $20 \times 20$ | 648,009 | 50,000 | 5 | 0.20 |
| | | 100,000 | 9 | 0.13 |
| $40 \times 40$ | 10,272,009 | 50,000 | 12 | 0.28 |
| | | 100,000 | 20 | 0.17 |
| $50 \times 50$ | 25,050,009 | 50,000 | 14 | 0.30 |
| | | 100,000 | 22 | 0.19 |

* Error was calculated over the temporal difference errors at the validation stage at epoch $t = 30$.

Whereas the size of the domain and the number of links in the network increase exponentially, the mean squared error increases very slowly and remains relatively small. Even on a commodity PC system, as described above, the computations require a reasonable amount of time. Note again that, after training, decision making is processed using the already trained networks that allow the application of the suggested techniques in online algorithms.

## 5. Discussion

In this paper, we consider the problem of detecting and tracking multiple hidden static and moving targets by a team of mobile agents and suggest two algorithms for agent navigation under uncertainty. In contrast to existing methods, the suggested algorithms effectively resolve uncertainty about the expected target's locations and navigate the agents in the presence of false-positive and false-negative detection errors.

The first algorithm, the Distributed EIG algorithm (DEIG), is a reactive online procedure in which each agent observes the environment and makes its decision using the obtained Expected Information Gain (EIG) value. This algorithm extends the previously developed greedy centralized EIG algorithm.

The DEIG algorithm uses Voronoi diagrams in parallel with the probability map. As a result, the search efforts are distributed in such a manner that the agents first consider their neighborhoods and then continue with detection in the other regions.

The main advantage of the DEIG algorithm is its computational simplicity and short run time. The conducted simulation studies demonstrate the algorithm's effectiveness, especially in cases with a low ratio of detection errors of the second type.

The second algorithm, the Collective Q-max (CQM) algorithm, includes deep Q-learning abilities that can be used both online and in the offline stage. The algorithm extends the previously developed Q-max algorithm for a team of agents.

The simulations of the algorithms were conducted using grids of sizes representing practical military and civil situations. A series of simulations show that both proposed algorithms outperform the known greedy and learning procedures and require reasonable and relatively moderate computation time. Note that, in the same situations, the existing algorithms for search either do not converge or require an extremely long computation time, which makes these algorithms unusable.

The CQM algorithm utilizes the learned information about the targets' location probabilities, the detection errors and the targets' motion. In the considered simulations, the targets' motion was characterized as 90% static (remaining stationary) and 10% as a random walk without any specific movement patterns. It is reasonable to expect that, if the targets move according to certain movement patterns, the CQM algorithm would be able to learn and detect these patterns, which can result in a shorter search time. Such scenarios, especially the case of the search game in which targets attempt to evade the search agents, require additional considerations.

The basic idea of the suggested learning procedure is the sequential training of the prediction neural network. Each agent is trained by its predecessor in the array of agents and trains its successor by simultaneously counting the expected actions. The training can be conducted both offline and online or can be processed in a mixed regime with online updating of the offline training results. In the considered scenarios, the order of the agents varies and is specified with respect to the distance of the agent from the starting point. Such a definition allows for the inclusion of all agents in minimal time in the search. In the other scenarios, the order can be predefined by the enumeration of the agents or can vary with respect to information measures or distances between the agents. Studies on different ordering schemes remain for further research.

The further development of the suggested algorithms will include their extension to detection in a domain with shadowing. For example, such tasks appear in the search in the terrain where certain areas are shadowed and exposed only during the agents' motion by certain trajectories.

This problem, as well as the search by marine on-water and underwater agents, gives rise to the detection problem with piecewise sensing, which requires combining the probability maps from the parts obtained by different agents at various times.

Finally, we plan to consider the influence of the expected information that is obtained by the agents in the next steps of the search and to determine the dependence of the discount factor used in the learning procedure with this information.

## 6. Conclusions

In this paper, we suggest two algorithms for detecting multiple static and moving targets by a team of mobile agents. The first algorithm is a reactive procedure, which implements the Voronoi diagrams, and the second algorithm is a procedure that implements deep Q-learning, which can be conducted both on- and offline.

Numerical simulations of the suggested algorithms in different scenarios of search for static and mobile targets demonstrate that the algorithm with deep Q-learning considerably outperforms the benchmark algorithms, including the algorithm based on Voronoi diagrams.

The main advantages of the algorithm are demonstrated in a noisy environment with the use of sensors with low sensitivity with many statistical errors of the first and second types.

The suggested algorithms can be used both for the further development of probabilistic search and detection methods and for practical applications for navigating autonomous drones and ground vehicles in the protection of facilities, smart city maintenance, mapping and surveying, precision agriculture, etc.

**Author Contributions:** Conceptualization, I.B.-G. and B.M.; methodology, B.M.; software, B.M.; formal analysis, B.M. and E.K.; investigation, B.M. and E.K.; writing—original draft preparation, B.M. and E.K.; writing—review and editing, I.B.-G.; supervision, I.B.-G. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Nahin, P.J. *Chases and Escapes: The Mathematics of Pursuit and Evasion*; Princeton University Press: Princeton, NJ, USA, 2007.
2. Koopman, B.O. *Search, and Screening*; Operation Evaluation Research Group Report, 56; Center for Naval Analysis: Rosslyn, VA, USA, 1946.
3. Stone, L.D. *Theory of Optimal Search*; Academic Press: New York, NY, USA, 1975.
4. Washburn, A.R. *Search and Detection*; ORSA Books: Arlington, VA, USA, 1989.
5. Stone, L.D.; Barlow, C.A.; Corwin, T.L. *Bayesian Multiple Target Tracking*; Artech House Inc.: Boston, MA, USA, 1999.
6. Kagan, E.; Ben-Gal, I. *Probabilistic Search for Tracking Targets*; Wiley & Sons: Chichester, UK, 2013.
7. Kagan, E.; Ben-Gal, I. *Search, and Foraging. Individual Motion and Swarm Dynamics*; CRC/Taylor & Francis: Boca Raton, FL, USA, 2015.
8. Stone, L.D.; Royset, J.O.; Washburn, A.R. *Optimal Search for Moving Targets*; Springer: Cham, Switzerland, 2016.
9. Senanayake, M.; Senthooran, I.; Barca, J.C.; Chung, H.; Kamruzzaman, J.; Murshed, M. Search and tracking algorithms for swarm of robots: A survey. *Robot. Auton. Syst.* **2016**, *75*, 422–434. [CrossRef]
10. Robin, C.; Lacroix, S. Multi-robot target detection and tracking: Taxonomy and survey. *Auton. Robot.* **2016**, *40*, 729–760. [CrossRef]
11. Ding, H. Models and Algorithms for Multiagent Search Problems. Ph.D. Thesis, Boston University, Boston, MA, USA, 2018.
12. Nguyen, T.T.; Nguyen, N.D.; Nahavand, S. Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *arXiv* **2019**, arXiv:1812.11794. Available online: https://arxiv.org/abs/1812.11794v2 (accessed on 7 May 2022). [CrossRef] [PubMed]
13. Dai, W.; Sartoretti, G. Multiagent search based on distributed deep reinforcement learning. In Proceedings of the 3rd Asian Conference Artificial Intelligence Technology (ACAIT 2019), Chongqing, China, 5–7 July 2019.
14. Jeong, H.; Hassani, H.; Morari, M.; Lee, D.D.; Pappas, G.J. Learning to Track Dynamic Targets in Partially Known Environments. *arXiv* **2020**, arXiv:2006.10190. Available online: https://arxiv.org/abs/2006.10190v1 (accessed on 7 May 2022).

15. Dell, R.F.; Eagle, J.N.; Martins, G.H.A.; Santo, A.G. Using multiple searchers in constrained-path, moving-target search problems. *Nav. Res. Logist.* **1996**, *43*, 463–480. [CrossRef]
16. Pack, D.J.; DeLima, P.; Toussaint, G.J.; York, G. Cooperative control of UAVs for localization of intermittently emitting mobile targets. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **2009**, *39*, 959–970. [CrossRef] [PubMed]
17. Matzliach, B.; Ben-Gal, I.; Kagan, E. Sensor fusion and decision-making in the cooperative search by mobile robots. In Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020), Valletta, Malta, 22–24 February 2020; pp. 119–126.
18. Matzliach, B.; Ben-Gal, I.; Kagan, E. Cooperative detection of multiple targets by the group of mobile agents. *Entropy* **2020**, *22*, 512. [CrossRef] [PubMed]
19. Elfes, A. Sonar-based real-world mapping, and navigation. *IEEE J. Robot. Autom.* **1987**, *3*, 249–265. [CrossRef]
20. Elfes, A. Occupancy grids: A stochastic spatial representation for active robot perception. In Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI1990), Cambridge, MA, USA, 27–29 July 1990; pp. 136–146.
21. Matzliach, B.; Ben-Gal, I.; Kagan, E. Detection of static and mobile targets by an autonomous agent with deep Q-learning abilities. *Entropy* **2022**, *8*, 1168. [CrossRef] [PubMed]
22. Dames, P.M. Distributed multi-agent search and tracking using the PHD filter. *Autonimous Robot.* **2020**, *44*, 673–689. [CrossRef]
23. Bertsekas, D. Multiagent Value Iteration Algorithms in Dynamic Programming and Reinforcement Learning. *arXiv* **2020**, arXiv:2005.01627. Available online: https://arxiv.org/abs/2005.01627v1 (accessed on 7 May 2022). [CrossRef]
24. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; Bradford Book, MIT Press: Cambridge, MA, USA, 1998.
25. Quiroga, F.; Hermosilla, G.; Farias, G.; Fabregas, E.; Montenegro, G. Position control of a mobile robot through deep reinforcement learning. *Appl. Sci.* **2022**, *12*, 7194. [CrossRef]
26. Brown, S. Optimal search for a moving target in discrete time and space. *Oper. Res.* **1980**, *28*, 1275–1289. [CrossRef]
27. Washburn, A.R. Search for a moving target: The FAB algorithm. *Oper. Res.* **1983**, *31*, 739–751. [CrossRef]