entropy

MDPI

Article

Detection of Static and Mobile Targets by an Autonomous Agent with Deep Q-Learning Abilities

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

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

Abstract: This paper addresses the problem of detecting multiple static and mobile targets by an autonomous mobile agent acting under uncertainty. It is assumed that the agent is able to detect targets at different distances and that the detection includes errors of the first and second types. The goal of the agent is to plan and follow a trajectory that results in the detection of the targets in a minimal time. The suggested solution implements the approach of deep Q-learning applied to maximize the cumulative information gain regarding the targets' locations and minimize the trajectory length on the map with a predefined detection probability. The Q-learning process is based on a neural network that receives the agent location and current probability map and results in the preferred move of the agent. The presented procedure is compared with the previously developed techniques of sequential decision making, and it is demonstrated that the suggested novel algorithm strongly outperforms the existing methods.

Keywords: search and detection; probabilistic decision-making; autonomous agent; deep Q-learning; neural network

Citation: Matzliach, B.; Ben-Gal, I.; Kagan, E. Detection of Static and Mobile Targets by an Autonomous Agent with Deep Q-Learning Abilities. *Entropy* **2022**, *24*, 1168. https://doi.org/10.3390/e24081168

Academic Editor: Gholamreza Anbarjafari

Received: 21 June 2022
Accepted: 11 August 2022
Published: 22 August 2022

1. Introduction

The detection of hidden stationary or moving targets is the first task of search procedures; this task focuses on recognizing target locations and precedes the chasing of the targets by the search agent [1,2]. Usually, the solution of the detection problem is represented by a certain distribution of the search effort over the considered domain [3,4]; for recent results and an overview of the progress in this field, see, e.g., [5–8].

In the simplest scenario of the detection of static targets by a static agent, it is assumed that the agent is equipped with a sensor that can obtain information (complete or incomplete) from all points in the domain. Using such a sensor, the agent screens the environment and accumulates information about the targets' locations; when the resulting accumulated information becomes sufficiently exact, the agent returns a map of the domain with the marked locations of the targets.

In the case of a moving agent, the detection process acts similarly, but it is assumed that the agent is able to move over the domain to clarify the obtained information or to reach a point from which the targets can be better recognized. A decision regarding the agent's movement is made at each step and leads the agent to follow the shortest trajectory to achieve the detection of all targets.

Finally, in the most complex scenario of moving target detection, the agent both moves within the domain to find a better observation position and tracks the targets to obtain exact information about each of their locations.

It is clear that in the first scenario, the agent has a passive role, and the problem is focused not on decision making, but on sensing and sensor fusion. However, in the case of a moving agent, the problem focuses on planning the agent's path.

In recent decades, several approaches have been suggested for planning the agent's motion and specifying decision-making techniques for detection tasks; for an overview of such research, see, e.g., [9,10]. Formally, such research addresses stochastic optimization methods that process offline and result in a complete agent trajectory or involve certain heuristic algorithms that allow the agent's path to be planned in real time.

In this research, we follow the direction of heuristic algorithms for search and detection with false positive and false negative detection errors [7,11–13] and consider the detection of static and moving targets. In addition, we assume that the agent is equipped with an on-board controller that is powerful enough to process deep Q-learning and train neural networks on relatively large data sets. Similar to previously obtained solutions [12,13], a data set is represented by an occupancy grid [14,15], and the decision making for the probability maps follows the Bayesian approach [7,8].

The implemented deep Q-learning scheme follows general deep learning techniques [16,17] applied to search and detection processes [18] and to navigation of mobile agents [19]. However, in addition to usual functionality, the suggested method utilizes the knowledge about the targets' locations in the form of probability map.

In the suggested algorithm, it is assumed that the agent starts with an initial probability map of the targets' locations and makes decisions about its further movements either by maximizing the expected cumulative information gain regarding the targets' locations or by minimizing the expected length of the agent's trajectory up to obtaining the desired probability map. For brevity, we refer to the first approach as the Q-max algorithm and the second approach as the Shortest Path Length (SPL) algorithm.

The maximization of the expected information gain and minimization of the expected path length are performed with a conventional dynamic programming approach, while the decision regarding the next step of the agent is obtained by the deep Q-learning of the appropriate neural network. As an input, the network receives the agent location and current probability map, and the output is the preferred move of the agent. The a priori training of the network is conducted on the basis of a set of simulated realizations of the considered detection process.

Thus, the main contributions of the paper are the following. In contrast to known search algorithms with learning, the suggested algorithm allows search and detection with false positive and false negative detection errors, and, in addition to general deep learning scheme, the suggested algorithm utilizes the current agent's knowledge about the targets' locations. Note that both featured of the suggested algorithm can be used for solving the other problems that can be formulated in the terms of autonomous agents and probability maps.

The algorithm and the training data set were implemented in the Python programming language with the PyTorch machine learning library. The performance of the algorithm was compared with the performance of previously developed methods. It was found that the novel deep Q-learning algorithm strongly outperforms (in the sense of obtaining the shortest agent path length) the existing algorithms with sequential decision-making and no learning ability. Therefore, it allows the targets to be detected in less time than the known methods.

2. Problem Formulation

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a finite set of cells that represents a gridded two-dimensional domain. It is assumed that in the domain C there are ξ targets, $\nu = 1, \ldots, \xi$, $\xi \leq n - 1$, which can stay in their locations or move over the set C, and an agent, which moves over the domain with the goal of detecting the targets.

It is assumed that the agent is equipped with an appropriate sensor such that the detection probability becomes higher as the agent moves closer to the target and as the agent observes the target location for a longer time, and the goal is to find a policy for the agent's motion such that it detects all ξ targets in minimal time or, equivalently, it follows the shortest trajectory.

This detection problem follows the Koopman framework of search and detection problems [3] (see also [4,8]) and continues the line of previously developed heuristic algorithms [12,13].

Following the occupancy grid approach [14,15], the state $s(c_i, t)$ of each cell $c_i \in C$, $i = 1, 2, \ldots, n$, at time $t = 1, 2, \ldots$ is considered a random variable with the values $s(c_i, t) \in \{0, 1\}$; $s(c_i, t) = 0$ means that the cell c_i at time t is empty, and $s(c_i, t) = 1$ means that this cell c_i at time t is occupied by a target. Since these two events are complementary, their probabilities satisfy

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

Detection is considered as an ability of the agent to recognize the states s_i of the cells, $i = 1, 2, \ldots, n$, and it is assumed that the probability of detecting the target is governed by the Koopman exponential random search formula [3]

$$Pr\{target\ detected\ in\ c_i \mid target\ located\ in\ c_i\} = 1 - exp\left[-\kappa\left(c_i, c_j, \tau\right)\right], \quad (2)$$

where $\kappa\left(c_i, c_j, \tau\right)$ is the search effort applied to cell c_i when the agent is located in cell c_j and the observation period is τ. Usually, the search effort $\kappa\left(c_i, c_j, \tau\right)$ is proportional to the ratio of observation period τ to the distance $d\left(c_i, c_j\right)$ between the cells c_i and c_j, $\kappa\left(c_i, c_j, \tau\right) \sim \tau / d\left(c_i, c_j\right)$, which represents the assumption that the shorter the distance $d\left(c_i, c_j\right)$ between the agent and the observed cell and the longer the observation period τ, the higher the detection probability is.

To define the possibility of false positive and false negative detection errors, we assume that the occupied cells, the states of which at time t, $t = 1, 2, \ldots$, are $s(c, t) = 1$, broadcast an alarm $\widetilde{a}(c, t) = 1$ with probability

$$p_{TA} = Pr\{\widetilde{a}(c, t) = 1 | s(c, t) = 1\}. \quad (3)$$

The empty cells, the states of which at time t, $t = 1, 2, \ldots$, are $s(c, t) = 0$, broadcast the alarm $\widetilde{a}(c, t) = 1$ with probability

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

where $0 \leq \alpha < 1$. The first alarm is called a true alarm, and the second alarm is called a false alarm.

By the Koopman formula, the probability of perceiving the alarms is

$$Pr\left\{alarm\ percieved\ at\ c_j \mid alarm\ sent\ from\ c_i\right\} = \exp\left[-d\left(c_i, c_j\right)/\lambda\right], \quad (5)$$

where λ is the sensitivity of the sensor installed on the agent; it is assumed that all the cells are observed during the same period, so the value τ can be omitted.

Denote by $p_i(t) = Pr\{s(c_i, t) = 1\}$ the probability that at time t, cell c_i is occupied by the target, that is, its state is $s(c_i, t) = 1$. The vector $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$ of probabilities $p_i(t)$, $i = 1, 2, \ldots, n$, also called the probability map of the domain, represents the agent's knowledge about the targets' locations in the cells $c_i \in C$, $i = 1, 2, \ldots, n$, at time t.

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

$$p\left(\widetilde{x}_j(c_i, t)\right) = p_i(t-1) p_{TA} \exp\left[-d\left(c_i, c_j\right)/\lambda\right] + \left(1 - p_i(t-1)\right) p_{FA} \exp\left[-d\left(c_i, c_j\right)/\lambda\right], \quad (6)$$

and the probability of the event $\overline{\widetilde{x}_j(c_i, t)}$, that this agent does not receive a signal at time t, is

$$p\left(\overline{\widetilde{x}_j(c_i, t)}\right) = 1 - p\left(\widetilde{x}_j(c_i, t)\right). \quad (7)$$

Note that the event $\widetilde{x}_j(c_i,t)$ represents the fact that the agent does not distinguish between true and false alarms, but it indicates that the agent receives a signal (which can be either a true or false alarm) from cell c_i. If $\alpha = 1$ and therefore $p_{TA} = p_{FA}$, then

$$p\left(\widetilde{x}_j(c_i,t)\right) = p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda\right], \tag{8}$$

which means that the agent's knowledge about the targets' locations does not depend on the probability map.

When the agent located in cell c_j receives a signal from cell c_i, the probability that cell c_i is occupied by the target is

$$Pr\left\{s(c_i,t)=1\middle|\widetilde{x}_j(c_i,t)\right\}=\frac{p_i(t-1)Pr\left\{\widetilde{x}_j(c_i,t)\middle|s(c_i,t)=1\right\}}{p_i(t-1)Pr\left\{\widetilde{x}_j(c_i,t)\middle|s(c_i,t)=1\right\}+(1-p_i(t-1))Pr\left\{\widetilde{x}_j(c_i,t)\middle|s(c_i,t)=0\right\}}, \tag{9}$$

and the probability that c_i is occupied by the target when the agent does not receive a signal from c_i is

$$Pr\left\{s(c_i,t)=1\middle|\overline{\widetilde{x}_j(c_i,t)}\right\}=\frac{p_i(t-1)Pr\left\{\overline{\widetilde{x}_j(c_i,t)}\middle|s(c_i,t)=1\right\}}{p_i(t-1)Pr\left\{\overline{\widetilde{x}_j(c_i,t)}\middle|s(c_i,t)=1\right\}+(1-p_i(t-1))Pr\left\{\overline{\widetilde{x}_j(c_i,t)}\middle|s(c_i,t)=0\right\}}, \tag{10}$$

where the probabilities $p_i(t-1)$, $i = 1, 2, \ldots, n$, represent the agent's knowledge about the targets' locations at time $t-1$ and it is assumed that the initial probabilities $p_i(0)$ at time $t = 0$ are defined with respect to prior information; if there is any initial information about the targets' locations, it is assumed that $p_i(0) = \frac{1}{2}$ for each $i = 1, 2, \ldots, n$.

In the framework of the Koopman approach, these probabilities are defined on the basis of Equations (6) and (7) and are represented as follows:

$$Pr\left\{s(c_i,t)=1\middle|\widetilde{x}_j(c_i,t)\right\}=\frac{p_i(t-1)p_{TA}}{p_i(t-1)p_{TA}+(1-p_i(t-1))\alpha p_{TA}}, \tag{11}$$

and

$$Pr\left\{s(c_i,t)=1\middle|\overline{\widetilde{x}_j(c_i,t)}\right\}=\frac{p_i(t-1)\left(1-p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda\right]\right)}{p_i(t-1)\left(1-p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda\right]\right)+(1-p_i(t-1))\left(1-\alpha p_{TA}\exp\left[-d\left(c_i,c_j\right)/\lambda\right]\right)}. \tag{12}$$

The defined process of updating the probabilities is illustrated in Figure 1.

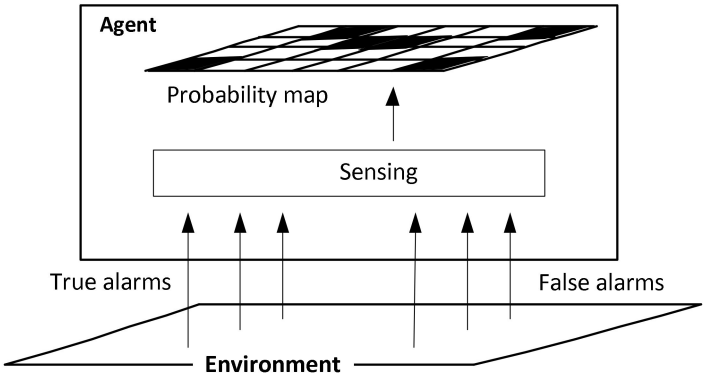


Figure 1. Receiving information and updating the probability map.

As illustrated by the figure, the agent receives true and false alarms through its on-board sensors, and based on this information, it updates the targets' location probabilities with Equations (11) and (12).

In the case of static targets, the location probabilities $p_i(t)$, $i = 1, 2, \ldots, n$, depend only on the agent's location at time t and its movements, while in the case of moving targets, these probabilities are defined both by the targets' and by the agent's activities. In the considered problem, it is assumed that the targets act independently on the agent's motion, while the agent is aware of the process that governs the targets' motion.

In general, the process of target detection is outlined as follows. At each step, the agent considers the probabilities $p_i(t)$, $i = 1, 2, \ldots, n$, for the targets' locations and makes a decision regarding its next movement. After moving to the new location (or remaining at its current location), the agent receives signals from the available cells and updates the probabilities $p_i(t)$ following Equations (11) and (12). The obtained updated probabilities $p_i(t+1)$ are used to continue the process.

The goal is to define the motion of the agent that results in the detection of all ξ targets in minimal time. As indicated above, in detecting the targets, the agent is not required to arrive to their exact locations, but it is required to specify the locations as definitively as possible. Since finding a general definition of the optimal agent's motion for any nontrivial scenario is computationally intractable, we are interested in a practically computable near-optimal solution.

3. Decision-Making Policy and Deep Q-Learning Solution

Formally, the detection problem of interest is specified as follows. Starting from the initial cell $c(0)$, at time t, the agent is located in cell $c(t)$ and makes a decision regarding its action $\eth(t) : C \to C$ that determines to which cell $c(t+1)$ the agent should move from its current location $c(t)$.

3.1. The Agent's Actions and Decision Making

We assume that the policy $\pi : C \times P \to \eth$ for choosing an action does not depend on time and is specified for any t by the current agent's location $c(t)$ and probability map $P(t)$. Then, the desired policy should produce actions such that the agent's trajectory from the cell $c(0)$ up to the final cell $c(T)$ is as short as possible (in the sense that the termination time T is as short as possible), and that by following this trajectory, the agent detects all ξ targets. It is assumed that the number ξ of targets is not available to the agent during detection and is used to indicate the end of the detection process.

With respect to the indicated properties of the desired agent trajectory, a search for the decision-making policy can follow either the maximization of the expected cumulative information gain over the trajectory or the direct optimization of the length of the trajectory in the indicated sense of minimal detection time. The first approach is referred to as the Q-max algorithm, and the second is referred to as the SPL algorithm.

In previous research [12,13], a similar search and detection problem is solved heuristically by evaluating the decisions made at each step of the search and detection process. In the first algorithm, the agent follows the maximal Expected Information Gain (EIG) over the cells that are reachable in a single step from the agent's current location; in the second algorithm, the agent moves one step toward the maximal expected information gain over all the cells, which is the Center Of View (COV) of the domain; and in the third algorithm, the agent moves toward the center of the distribution or the Center Of Gravity (COG) with respect to the current probability map.

In this paper, we address a more sophisticated approach that implements deep Q-learning techniques. First, we consider the information-based Q-max algorithm and then the SPL algorithm.

Let us start with the Q-max solution of the considered detection problem. Assume that at each time t the agent is located in cell $c(t)$ and action $\eth(t)$ is chosen from among the possible movements from cell $c(t)$, which are to step "forward", "right-forward", "right",

"right-backward", "backward", "left-backward", "left", or "left-forward" or "stay in the current cell". Symbolically, we write this choice as

$$\eth(t) \in \mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}. \tag{13}$$

Denote by $P_{\eth}(t+1)$ a probability map that should represent the targets' locations at time $t+1$ given that at time t, the agent chooses action $\eth(t)$. Then, given action $\eth_t$, the immediate expected informational reward of the agent is defined as

$$R(\eth, t) = D_{KL}(P_{\eth}(t+1)||P(t)), \tag{14}$$

that is, the Kullback–Leibler distance between the map $P_{\eth}(t+1)$ and the current probability map $P(t)$. Given a policy π for choosing an action, the expected cumulative discounted reward obtained by an agent that starts in cell $c(t)$ with probability map $P(t)$ and chooses action $\eth(t)$ is

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

where as usual, the discount factor is $0 < \gamma \leq 1$, and the goal is to find a maximum value

$$Q(c(t), P(t), \eth(t)) = \max_{\pi} q_\pi(c(t), P(t), \eth(t)) \tag{16}$$

of the expected reward q_π over all possible policies π that can be applied after action $\eth(t)$ is chosen at time t.

Since the number of possible policies is infinite, the value $Q(c(t), P(t), \eth(t))$ of the maximal expected cumulative discounted reward cannot be calculated exactly, and for any realistic scenario, it should be approximated. Below, we follow the deep Q-learning approach and present the Q-max algorithm, which approximates the values $Q(c(t), P(t), \eth(t))$ of the reward for all possible actions (13) and therefore provides criteria for choosing the actions.

3.2. Dynamic Programming Scheme with Prediction and Target Neural Networks

The learning stage in the suggested Q-max algorithm is based on a neural network with dynamic programming for predicting current rewards. In the simplest configuration, which is still rather effective, the network consists of one input layer, which includes $2n$ neurons (recall that n is the size of the domain); one hidden layer, which also includes $2n$. neurons; and an output layer, which includes $\#\mathbb{A} = 9$ neurons with respect to the number of possible actions (13).

The inputs of the network are as follows. The first chunk of n inputs $(1, 2, \ldots n)$ receives a binary vector that represents the agent location; namely, if the agent is located in cell c_j, then the jth input of the network is equal to 1 and the other $n-1$ inputs are equal to 0. The second chunk of n inputs $(n+1, n+2, \ldots 2n)$ receives the target location probabilities; namely, the $(n+i)$th input receives the target location probability p_i, $i = 1, 2, \ldots, n$, as it appears in the probability map P.

The hidden layer of the network consists of $2n$ neurons, each of which implements a fully connected linear layer and sigmoid activation function $f(x) = 1/(1+e^{-x})$. This activation function was chosen from among several possible activation functions, such as the step function, Softplus function and SiLU function, and it was found that it provides adequate learning in all conducted simulations.

The nine neurons of the output layer correspond to the possible actions. Namely, the first output corresponds to the action $\eth_1 =$ " $\uparrow$ ", "step forward"; the second output corresponds to the action $\eth_2 =$ " $\nearrow$ ", "step right-forward"; and so on up to action $\eth_9 =$ " $\odot$ ", which is "stay in the current cell". The value of the kth output is the maximal expected cumulative discounted reward $Q(c_j, P, \eth_k)$ obtained by the agent if it is in cell c_j, $j = 1, 2, \ldots, n$, and given the target location probabilities $P = (p_1, p_2, \ldots, p_n)$, it chooses action $\eth_k$, $k = 1, 2, \ldots, 9$.

The scheme of the network is shown in Figure 2.

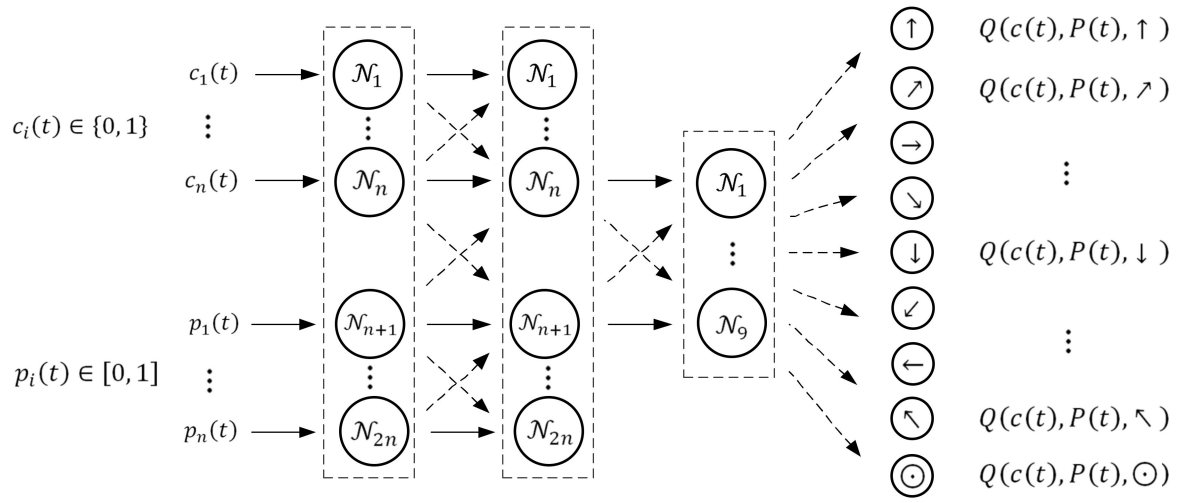


Figure 2. The neural network scheme used in the learning stage of the Q-max algorithm.

The training stage of the network implements deep Q-learning techniques, which follow the dynamic programming approach (see, e.g., [16,17]). In general, the Bellman equation for calculating the maximal cumulative discounted reward is as follows:

$$Q(c(t), P(t), \eth(t)) = R(\eth, t) + \gamma \max_{\eth \in \mathbb{A}} Q(c(t+1), P(t+1), \eth), \tag{17}$$

and this equation forms a basis for updating the weights of the links in the network. The data flow specified by this equation is shown in Figure 3.

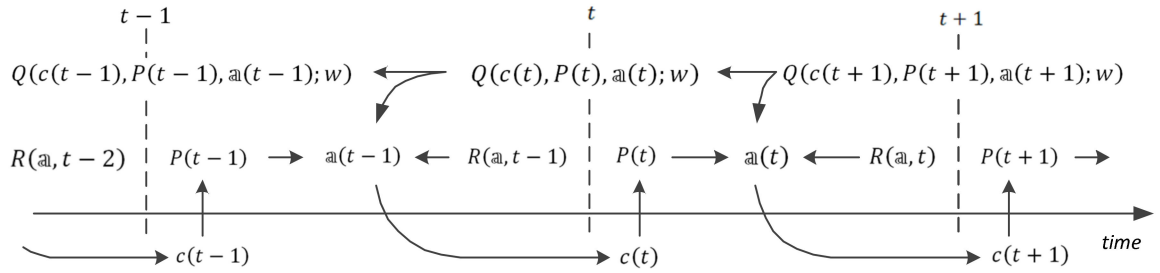


Figure 3. Scheme of the data flow in the training stage of the network.

Let w be a vector of the link weights of the network. In the considered case, there are $4n^2 + 18n + 2n + 9$ values of the weights, where $4n^2$ is the number of links between the input layers and the hidden layer, $18n$ is the number of links between the hidden layer and the output layer, $2n$ is the number of biases in the hidden layer and 9 is the number of biases in the output layer.

In addition, to distinguish these steps and to separate them from the real time, we enumerate the training steps by $l = 1, 2, \ldots$ below and retain the symbol t for the real-time moments of the detection process.

Denote by $Q(c(l), P(l), \eth(l); w)$ the maximal cumulative discounted reward calculated at step $l = 1, 2, \ldots$ by the network with weights w, and denote by $Q^+(c(l), P(l), \eth(l); w')$ the expected maximal cumulative discounted reward calculated using the vector w' of the updated weights following the recurrent Equation (17); that is,

$$Q^+\big(c(l), P(l), \eth(l); w'\big) = R(\eth, l) + \gamma \max_{\eth \in \mathbb{A}} Q\big(c(l+1), P(l+1), \eth; w'\big). \tag{18}$$

Then, the temporal difference learning error is

$$\Delta_l(Q) = Q^+\left(c(l), P(l), \ni(l); w'\right) - Q(c(l), P(l), \ni(l); w). \tag{19}$$

In practice, the values $Q(c(l), P(l), \ni(l); w)$ and $Q^+(c(l), P(l), \ni(l); w')$ are associated with separate neural networks; the first is called the prediction network, and the second is called the target network.

3.3. Model-Free and Model-Based Learning

The updating of the weights w in the prediction network is conducted by following basic backpropagation techniques; namely, the weights for the next step $l+1$ are updated with respect to the temporal difference learning error $\Delta_l(Q)$ calculated at the current step l. The weights w' in the target network are updated to the values of the weights w after an arbitrary number of iterations. In the simulations presented in subsequent sections, such updating was conducted at every fifth step.

The presented training procedure directly uses the events occurring in the environment and does not require prior knowledge about the targets' abilities. In other words, by following this procedure, the agent detects the targets in the environment and simultaneously learns the environment and trains the neural network that supports the agent's decision-making processes. We refer to such a scenario as model-free learning.

The actions of the presented online model-free learning procedure are illustrated in Figure 4.

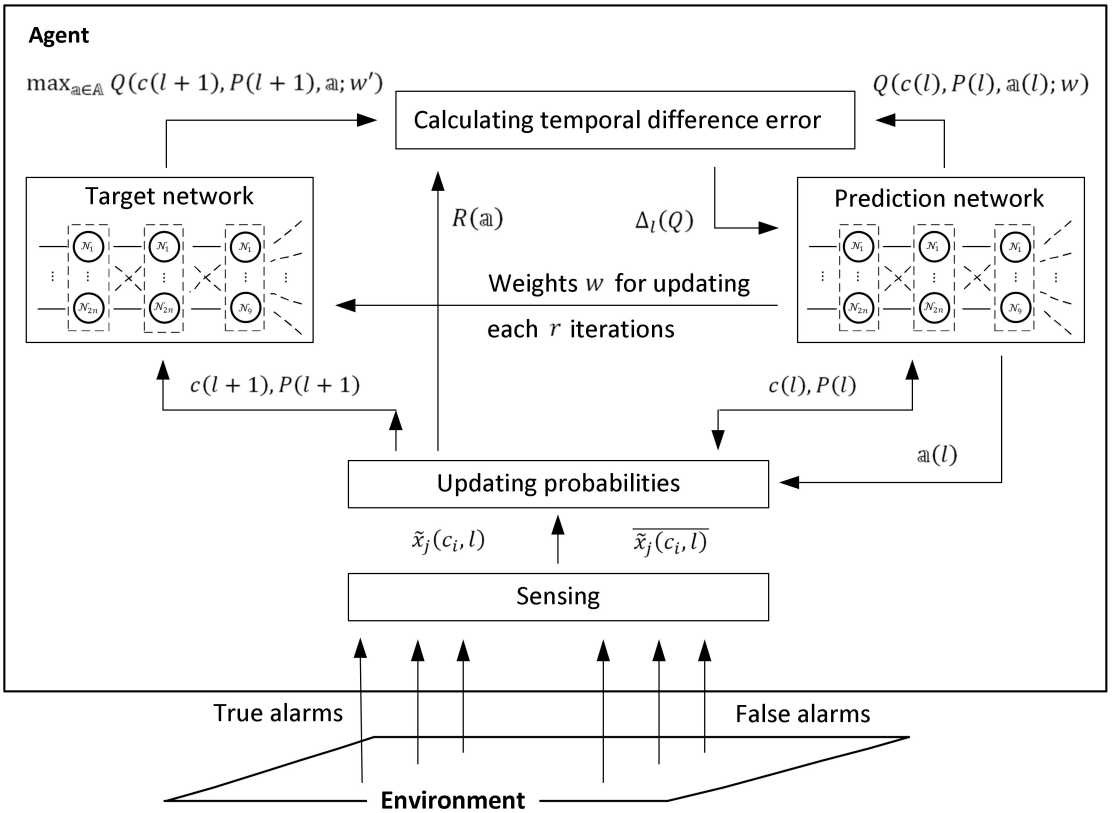


Figure 4. The actions of the online model-free learning procedure of the Q-max algorithm.

Following the figure, at step l, the target location probabilities $Pr\{s(c_i, l) = 1 | \tilde{x}_j(c_i, l)\}$ and $Pr\left\{s(c_i, l) = 1 \middle| \overline{\tilde{x}_j(c_i, l)}\right\}$, i,j $= 1, 2, \ldots,$ n, are updated by Equations (11) and (12) with respect to the events $\tilde{x}_j(c_i, l)$ and $\overline{\tilde{x}_j(c_i, l)}$ of receiving and not receiving a signal from cell c_i while the agent is in cell c_j. The updated target location probabilities are used for calculating the value of the immediate reward $R(\ni, l)$ by Equation (14) and

the value $Q^+(c(l), P(l), \eth(l); w')$ by Equation (18) in the target network. In parallel, the value $Q(c(l), P(l), \eth(l); w)$ of the prediction network is used for choosing the action and consequently for specifying the expected position of the agent in the environment. After calculating the temporal difference error $\Delta_l(Q)$ between the Q-values in the target and in the prediction network by Equation (19), the weights w in the prediction network are updated, and the process continues with step $l+1$.

Note that in all the above definitions, the cumulative reward does not depend on the previous trajectory of the agent. Hence, the process that governs the agent's activity is a Markov process with states that include the positions of the agent and the corresponding probability maps. This property allows the use of an additional offline learning procedure based on the knowledge of the targets' abilities.

Namely, if the abilities of the targets are known and can be represented in the form of transition probability matrices that govern the targets' motion, the learning process can be conducted offline without checking the events occurring in the environment. In this scenario, at step l, instead of the target location probabilities $Pr\{s(c_i, l) = 1 | \widetilde{x}_j(c_i, l)\}$ and $Pr\left\{s(c_i, l) = 1 | \overline{\widetilde{x}_j(c_i, l)}\right\}$, i, j $= 1, 2, \ldots,$ n, 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 at the previous step $l-1$.

Based on the previous definitions, these probabilities are defined as follows:

$$\begin{aligned} &Pr\{s(c_i, l) = 1 | s(c_i, l-1) = 1\} \\ &= \frac{p_i(l-1) p_{TA} \exp\left[-d\left(c_i, c_j\right)/\lambda\right]}{p_i(l-1)(1-\alpha)+\alpha} + \frac{p_i(l-1)\left(1-\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)^2}{p_i(l-1)\left(1-p_{TA}\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)+(1-p_i(l-1))\left(1-\alpha p_{TA}\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)}, \end{aligned} \tag{20}$$

and

$$\begin{aligned} &Pr\{s(c_i, l) = 1 | s(c_i, l-1) = 0\} \\ &= \frac{p_i(l-1)\alpha p_{TA} \exp\left[-d\left(c_i, c_j\right)/\lambda\right]}{p_i(l-1)(1-\alpha)+\alpha} + \frac{p_i(l-1)\left(1-\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)\left(1-\alpha p_{TA}\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)}{p_i(l-1)\left(1-p_{TA}\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)+(1-p_i(l-1))\left(1-\alpha p_{TA}\exp\left[-d\left(c_i, c_j\right)/\lambda\right]\right)}. \end{aligned} \tag{21}$$

Since the presented procedure is based on certain knowledge about the targets' activity, it is called the model-based learning procedure.

The actions of the model-based offline learning procedure are illustrated in Figure 5.

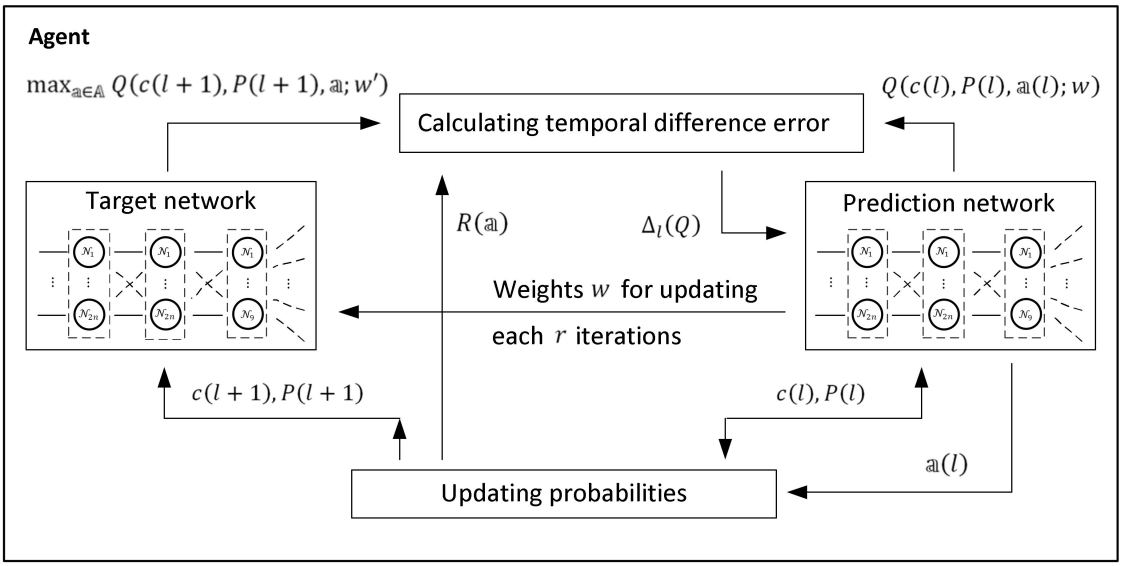


Figure 5. The actions of the offline model-based learning procedure of the Q-max algorithm.

The model-based learning procedure differs from the model-free learning procedure only in the use of the target location probabilities and the method of updating them. In the model-free procedure, these probabilities are specified based on the events occurring in the

environment. In the model-free procedure, they are calculated by following the Markov property of the system without referring to the real events in the environment.

As a result, in the model-free procedure, the learning is slower than in the model-based procedure. However, while in the first case, the agent learns during the detection process and can act without any prior information about the targets' abilities, in the second case, it starts detection only after offline learning and requires an exact model of the targets' activity. Thus, the choice of a particular procedure is based on the considered practical task and available information.

3.4. The Choice of the Actions at the Learning Stage

As indicated above, given the agent's location $c(l)$ and the targets' probability map $P(l)$, the neural networks used at the learning stage provide *nine* output Q-values that are associated with possible actions $\eth_k \in \mathbb{A}$, $k = 1, 2, \ldots, 9$, namely,

$$Q(c(l), P(l), \eth_1; w),\ Q(c(l), P(l), \eth_2; w), \ldots, Q(c(l), P(l), \eth_9; w),$$

where $\eth_1 =$ " $\uparrow$ " ("step forward"), $\eth_2 =$ " $\nearrow$ " ("step right-forward"), and so on up to $\eth_9 =$ " $\odot$ " ("stay in the current cell").

The choice among the actions $\eth_k \in \mathbb{A}$ is based on the corresponding Q-values $Q(c(l), P(l), \eth_k; w)$, $k = 1, 2, \ldots, 9$, and implements exploration and exploitation techniques. At the initial step $l = 0$, when the agent has no prior learned information about the targets' locations, action $\eth(l) \in \mathbb{A}$ is chosen randomly. Then, after processing the step prescribed by action $\eth(l)$, the next action $\eth(l+1)$ is chosen either on the basis of the target location probabilities learned by the neural networks or randomly from among the actions available at this step. The ratio of random choices decreases with the number of steps, and after finalizing the learning processes in the neural networks, the actions are chosen with respect to the Q-values only.

Formally, this process can be defined using different policies, for example, with a decaying ϵ-greedy policy that uses the probability ϵ, which decreases with the increase in the number of steps from its maximal value $\epsilon = 1$ to the minimal value $\epsilon = 0$. The agent chooses an action randomly with probability ϵ and according to the greedy rule $\underset{\eth \in \mathbb{A}}{\operatorname{argmax}} Q(c(l), P(l), \eth; w)$ with probability $1 - \epsilon$. In this policy, the choice of the action is governed by the probability ϵ and does not depend on the Q-values of the actions.

A more sophisticated policy of intermittence between random and greedy choice is the SoftMax policy. In this policy, the probability $p(\eth_k | Q; \eta)$ of choosing action $\eth_k$ is defined with respect to both the parameter $\eta \in [0, +\infty)$ and the Q-values of the actions:

$$p(\eth_k | Q; \eta) = \frac{\exp[Q(c(l), P(l), \eth_k; w)/\eta]}{\sum_{j=1}^{9} \exp\left[Q\left(c(l), P(l), \eth_j; w\right)/\eta\right]}. \tag{22}$$

Therefore, if $\eta \to 0$, then $p(\eth_k | Q; \eta) \to 1$ for $\eth_k = \underset{\eth \in \mathbb{A}}{\operatorname{argmax}} Q(c(l), P(l), \eth; w)$ and $p(\eth_k | Q; \eta) \to 0$ for all other actions, and if $\eta \to \infty$, then $p(\eth_k | Q; \eta) \to \frac{1}{9}$, which corresponds to a randomly chosen action. The intermediate values $0 < \eta < \infty$ correspond to the probabilities $p(\eth | Q; \eta) \in (0, 1)$ and govern the randomness of the action choice. In other words, the value of the parameter η decreases with the increasing number of steps l from its maximal value to zero; thus, for the unlearned networks, the agent chooses actions randomly and then follows 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. In the simulations, we considered both policies and finally implemented the SoftMax policy since it provides more correct choices, especially in cases with relatively high Q-values associated with different actions.

3.5. Outline of the Q-Max Algorithm

Recall that according to the formulation of the detection problem, the agent acts in the finite two-dimensional domain $C = \{c_1, c_2, \ldots, c_n\}$ and moves over this domain with the aim of detecting $\xi \leq n-1$ hidden targets. At time $t = 0, 1, 2, \ldots$ in cell $c(t) \in C$, the agent observes the domain (or, more precisely, screens the domain with the available sensors) and creates the probability map $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$, where $p_i(t)$ is the probability that at time t, cell c_i is occupied by a target, $i = 1, 2, \ldots, n$. Based on the probability map $P(t)$, the agent chooses an action $\eth(t) \in \mathbb{A}$. By processing the chosen action $\eth(t)$, the agent moves to the next cell $c(t+1)$, and this process continues until the targets' locations are detected. The agent's goal is to find a policy of choosing the action that provides the fastest possible detection of all the targets with a predefined accuracy.

In contrast to the recently suggested algorithms [12,13], which directly implement one-step decision making, the presented novel algorithm includes learning processes and can be used both with model-free learning for direct online detection and with model-based learning for offline policy planning and further online applications of this policy. Since both learning processes follow the same steps (with the only difference being in the source of information regarding the targets' location probabilities), below, we outline the Q-max algorithm with model-based learning.

The Q-max algorithm with model-based learning includes three stages: in the first stage, the algorithm generates the training data set, which includes reasonable probability maps with possible agent locations; in the second stage, it trains the prediction neural network using the generated data set; and in the third stage, the algorithm solves the detection problem by following the decisions made by the trained prediction neural network. Algorithm 1 outlines the first stage that is generating of the training data set.

Algorithm 1. Generating the training data set

Input: domain $C = \{c_1, c_2, \ldots, c_n\}$,
set $\mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}$ of possible actions,
probability p_{TA} of true alarms (Equation (3)),
rate α of false alarms and their probability $p_{FA} = \alpha p_{TA}$ (Equation (4)),
sensor sensitivity λ,
range $[\xi_1, \xi_2]$ of possible numbers $0 < \xi_1 < \xi_2 \leq n-1$ of targets,
length $L \in (0, \infty)$ of the agent's trajectory,
number $N \in (0, \infty)$ of agent trajectories,
initial probability map $P(0)$ on the domain C.
Output: data set that is an $L \times N$ table of pairs (c, P) of agent positions c and corresponding probability maps P.
1. Create the $L \times N$ data table.
2. For each agent trajectory $j = 1, \ldots, N$ do:
3. Choose a number $\xi \in [\xi_1, \xi_2]$ of targets according to a uniform distribution on the interval $[\xi_1, \xi_2]$.
4. Choose the target locations $c_1, c_2, \ldots, c_\xi \in C$ randomly according to the uniform distribution on the domain C.
5. Choose the initial agent position $c(0) \in C$ randomly according to the uniform distribution on the domain C.
6. For $l = 0, \ldots, L-1$ do:
7. Save the pair $\langle c(l), P(l) \rangle$ as the jth element of the data table.
8. Choose an action $\eth(l) \in \mathbb{A}$ randomly according to the uniform distribution on the set $\mathbb{A}$.
9. Apply the chosen action and set the next position $c(l+1) = \eth(c(l))$ of the agent.
10. Calculate the next probability map $P(l+1)$ with Equations (20) and (21).
11. End for
12. End for
13. Return the data table.

The data training data set includes N random trajectories of length L. Each element of the data set is a pair of an agent position and a probability map.

The reason for generating the data instead of drawing it randomly is that the training data set is used at the learning stage of the prediction network, so it should represent the data in as realistic a form as possible. Since in the generated data set, the agent's positions are taken from the connected trajectory and the corresponding probability maps are calculated with respect to these positions, possible actions, sensing abilities and environmental conditions, it can be considered a good imitation of real data.

The generated agent positions and corresponding probability maps are used as an input of the prediction neural network in the training stage. The goal of the training is specified by the objective probability map $P^* = \{p_1^*, p_2^*, \ldots, p_n^*\}$, which defines the target location probabilities that provide sufficient information for the immediate detection of all the targets. In the best case, we have probabilities $p_i^* \in \{0, 1\}$, and in practical scenarios, it is assumed that either $p_i^* \in [0, \varepsilon]$ or $p_i^* \in [1 - \varepsilon, 1]$ for certain $0 < \varepsilon \ll 1$, $i = 1, 2, \ldots, n$.

The training stage of the Q-max algorithm is implemented in the form of Algorithm 2, which is outlined below (the scheme of the learning procedure is shown in Figure 5).

Algorithm 2. Training the prediction neural network

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:
sigmoid function $f(x) = 1/(1 + e^{-x})$.
Loss function:
mean square error (MSE) function.
Input: domain $C = \{c_1, c_2, \ldots, c_n\}$,
set $\mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}$ of possible actions,
probability p_{TA} of true alarms (Equation (3)),
rate α of false alarms and their probability $p_{FA} = \alpha p_{TA}$ (Equation (4)),
sensor sensitivity λ,
discount factor γ,
objective probability map P^* (obtained by using the value ε),
number r of iterations for updating the weights,
initial value η (Equation (22)) and its discount factor δ,
learning rate ρ (with respect to the type of optimizer),
number M of epochs,
initial weights w of the prediction network and initial weights $w' = w$ of the target network,
training data set (that is, the $L \times N$ table of (c, P) pairs created by Procedure 1).
Output: The trained prediction network.
1. Create the prediction network.
2. Create the target network as a copy of the prediction network.
3. For each epoch $j = 1, \ldots, M$ do:
4. For each pair (c, P) from the training data set, do:
5. For each action $\eth \in \mathbb{A}$ do:
6. Calculate the value $Q(c, P, \eth; w)$ with the prediction network.
7. Calculate the probability $p(\eth|Q; \eta)$ (Equation (22)).
8. End for.
9. Choose an action according to the probabilities $p(\eth|Q; \eta)$.
10. Apply the chosen action and set the next position $c' = \eth(c)$ of the agent.
11. Calculate the next probability map P' with Equations (20) and (21).
12. If $P = P^*$ or $c' \notin C$, then
13. Set the immediate reward $R(\eth) = 0$.

14. Else
15. Calculate the immediate reward $R(\eth)$ with respect to P and P' (Equation (14)).
16. End if.
17. For each action $\eth \in \mathbb{A}$ do:
18. If $P = P^*$ then
19. Set $Q(c', P', \eth; w') = 0$.
20. Else
21. Calculate the value $Q(c', P', \eth; w')$ with the target network.
22. End if.
23. End for.
24. Calculate the target value $Q^+ = R(\eth) + \gamma \max_{\eth \in \mathbb{A}} Q(c', P', \eth; w')$ (Equation (17)).
25. Calculate the temporal difference learning error as $\Delta_l(Q) = Q^+ - Q(c, P, \eth; w)$ for the chosen action $\eth$ (Equation (19)) and set $\Delta_l(Q) = 0$ for all other actions.
26. Update the weights w in the prediction network by backpropagation with respect to the error $\Delta_l(Q)$.
27. Every r iterations, set the weights of the target network as $w' = w$.
28. End for.

The validation of the network was conducted on a validation data set that includes the pairs (c, P), which are similar to the pairs appearing in the training data set but were not used in the training procedure; the size of the validation data set is approximately ten percent of the size of the training data set.

After training, the Q-max algorithm can be applied to simulated data or in a real search over a domain. It is clear that the structure of the algorithm mimics the search conducted by rescue and military services: first, the algorithm learns the environment (by itself or at least by using the model) and then continues with the search in the real environment, where the probability map is updated with respect to the received alarms and acquired events (Equations (11) and (12)) and decision-making is conducted using the prediction network.

3.6. The SPL Algorithm

Now let us consider the SPL algorithm. Formally, it follows the same ideas and implements the same approach as the Q-max algorithm, but it differs in the definition of the goal function. In the SPL algorithm, the goal function directly represents the aim of the agent to detect all the targets in a minimal number of steps or to take a minimal number of actions before reaching the termination condition.

In parallel to the reward $R(\eth, t)$ defined by Equation (14) for action $\eth \in \mathbb{A}$ conducted at time t, we define the penalty or the price paid by the agent for action $\eth \in \mathbb{A}$ at time t. In the case of the shortest path length, the payoff represents the steps of the agent; that is,

$$O(\eth, t) = 1 \tag{23}$$

for each time $t = 1, 2, \ldots$ until termination of the search. Note again that even if the agent chooses to stay at its current position, the payoff is calculated as 1.

Then, given a policy π for choosing an action, the expected cumulative payoff of an agent that starts in cell $c(t)$ with probability map $P(t)$ and chooses action $\eth(t)$ is

$$pl_\pi(c(t), P(t), \eth(t)) = \mathbb{E}_\pi\Big\{\sum_{\tau=0}^{\infty} O(\eth, t+\tau)\Big\}, \tag{24}$$

and the goal is to find the minimum value

$$SPL(c(t), P(t), \eth(t)) = \min_\pi pl_\pi(c(t), P(t), \eth(t)) \tag{25}$$

of the expected payoff pl_π over all possible policies π that can be applied after action $\eth(t)$ is chosen at time t.

Then, the Bellman equation for calculating the defined minimal expected path length is

$$SPL(c(t), P(t), \eth(t)) = O(\eth, t) + \min_{\eth \in \mathbb{A}} SPL(c(t+1), P(t+1), \eth), \tag{26}$$

and the equations that define the training and functionality of the neural networks follow this equation and have the same form as in the Q-max algorithm (with the obvious substitution of maximization by minimization and the use of $\gamma = 1$).

4. Simulation Results

The suggested algorithm was implemented and tested in several scenarios and its functionality was compared with the functionality of previously developed heuristic algorithms and, in certain simple setups, with the algorithm that provides optimal solution. Numerical simulations include training of neural network, simulation of the detection process by Q-max and SPL algorithms and their comparisons with heuristic and optimal solutions.

Numerical simulations were implemented using basic tools of the Python programming language with the PyTorch machine learning library, and the trials were run on a PC Intel® Core™ i7-10700 CPU with 16 GB RAM. In the simulations, the detection was conducted over a gridded square domain of size $n = n_x \times n_y$ cells, and it was assumed that the agent and each target could occupy only one cell. Given this equipment, we measured the run time of the simulations for different datasets, which demonstrated that the suggested algorithms are implementable on usual computers and do not require specific apparats for their functionality.

4.1. Network Training in the Q-Max Algorithm

First, let us consider the simulation of the network training. The purpose of these simulations is to verify the training method and demonstrate a decrease in the temporal difference learning error $\Delta(Q)$ with an increasing number of learning epochs. Since the network training is the same for both the Q-max and SPL algorithms, we consider the training for the Q-max algorithm.

The training data set was generated using the parameters $n = 10 \times 10 = 100$, $p_{TA} = 1$, $\alpha = 0.5$, $\lambda = 15$, $\xi_1 = 1$, $\xi_2 = 10$, $L = 50$, $N = 200$ and $p_i(0) = 0.05$, $i = 1, 2, \ldots, n$. The size of the training data set was 10000.

The input parameters in the simulation used the same values of $n = 10 \times 10 = 100$, $p_{TA} = 1$, $\alpha = 0.5$, and $\lambda = 15$, and we also specified $\gamma = 0.9$ and P^* with $\varepsilon = 0.05$, $r = 5$, $\eta = 100000$, $\delta = 0.99$ and $\rho = 0.001$. The number of epochs in the simulation was $M = 30$.

The initial weights w were generated by the corresponding procedures of the PyTorch library. The optimizer used in the simulation was the ADAM optimizer from the PyTorch library.

The average time required for training the prediction neural network was approximately 10 min (on the PC described above), which is a practically reasonable time for an offline procedure. Note that after offline training, online decision-making is conducted directly by immediate choice without additional calculations.

The results of the simulations are shown in Figure 6. The presented graph was obtained by averaging the temporal difference learning errors over 10,000 pairs in the data set.

The temporal difference learning error decreases both in the training stage and in the validation stage of the learning process, and the smoothed graphs for both stages are exponential graphs with similar rates of decrease. This validates the effectiveness of the learning process and shows that progress in the network training leads to better processing of previously unknown data from the validation data set.

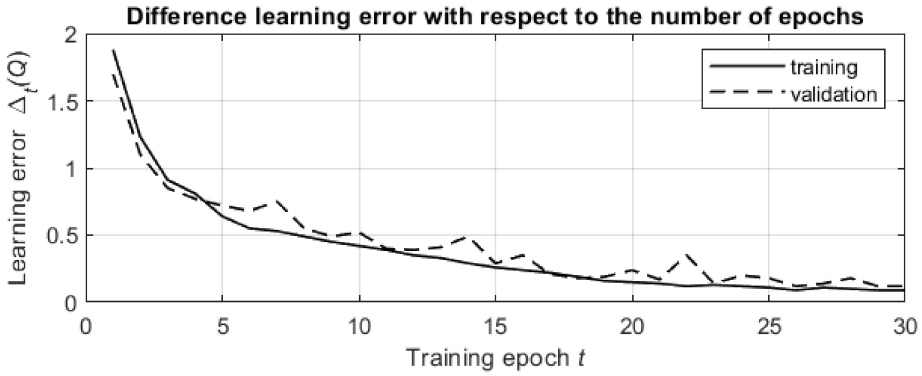


Figure 6. The change in the temporal difference learning error with respect to the number of training epochs. The solid line is associated with the training stage, and the dashed line is associated with the validation stage.

4.2. Detection by the Q-Max and SPL Algorithms

In the next simulations, we considered the detection process with the proposed Q-max and SPL algorithms and compared both algorithms with random detection, which provides the lower bound of the cumulative reward (for the Q-max algorithm) and payoff (for the SPL algorithm).

Both algorithms used the same neural network as above, and the random detection process was initialized with the same parameters as above. However, for better comparison, in the simulations of both algorithms and of random detection, we used the same number of targets $\xi = 2$, which were located at the points $(5, 0)$ and $(0, 9)$, and the initial position of the agent was $c(0) = (9, 4)$. By choosing these positions of the targets and the agent, it is easy to demonstrate (a) the difference between the search processes (in which the agent first moves to the closer target and then to the distant target) and the detection process (in which the agent moves to the point that provides the best observation of both targets) and (b) the motion of the agent over the domain to maximize the immediate reward or minimize the immediate payoff.

The results of the simulations are shown in Figure 7. Figure 7a shows the discounted cumulative reward for the Q-max algorithm in comparison with that of the random detection process, and Figure 7b shows similar graphs for the SPL algorithm and the random detection process.

The detection by the proposed algorithms is much better than the detection by the random procedure. Namely, the Q-max algorithm results in 20.5 units of discounted cumulative reward, while the random procedure achieves only 13.4 units of discounted reward in the same number of steps. In other words, the Q-max algorithm is nearly 1.5 times more effective than the random procedure. Similarly, while the random procedure requires 40 steps to detect the targets, the SPL algorithm requires only 20 steps, which means that the SPL algorithm is 50% better than the random procedure.

From these comparisons, it follows that the suggested algorithms outperform the random procedure in terms of both the informational reward and the agent's path length. However, as follows from the next simulations, the numbers of agent actions up to termination in the Q-max and SPL algorithms are statistically equal, allowing either algorithm to be applied with respect to the considered practical task.

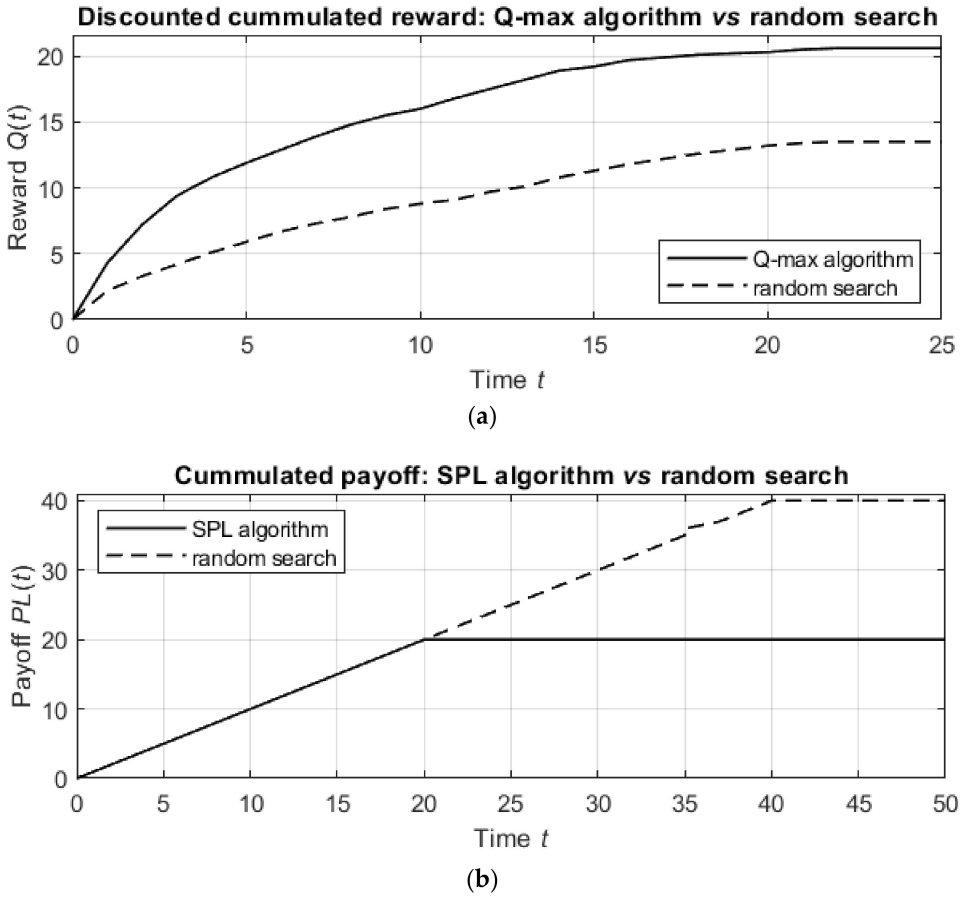


Figure 7. Discounted cumulative reward of detection by the Q-max algorithm (**a**) and cumulative payoff of detection by the SPL algorithm (**b**) compared with the results obtained by the random detection procedure. The solid line in both figures is associated with the suggested algorithms (Q-max and SPL), and the dashed line is associated with the random choice of actions.

4.3. Comparison between the Q-Max and SPL Algorithms and the Eig, Cov and Cog Algorithms

The third set of simulations included comparisons of the suggested Q-max and SPL algorithms with the previously developed heuristic methods [12,13], which implement one-step optimization.

The simplest algorithm is based on the expected information gain, which is an immediate expected information reward

$$EIG(\eth, t) = R(\eth, t) = D_{KL}(P_{\eth}(t+1)||P(t)). \tag{27}$$

as above, here, $P_{\eth}(t+1)$ stands for the probability map that is expected to represent the targets' locations at time $t+1$ given that at time t, the agent chooses action $\eth(t) \in \mathbb{A}$ and $P(t)$ is the current probability map. Then, the next action is chosen as

$$\eth(t+1) = \underset{\eth \in \mathbb{A}}{\operatorname{argmax}} EIG(\eth, t). \tag{28}$$

A more sophisticated algorithm addresses the center of view, which is defined as the cell in which the agent can obtain the maximal expected information gain

$$COV(t) = \underset{c \in \mathbb{C}}{\operatorname{argmax}} D_{KL}(P_c(t+1)||P(t)), \tag{29}$$

where $P_c(t+1)$ is a probability map that is expected to represent the targets' locations at time $t+1$ when the agent is located in cell c. Then, the next action is chosen as

$$\eth(t+1) = \underset{\eth \in \mathbb{A}}{\operatorname{argmin}}\, d(COV(t), \eth(c(t))), \tag{30}$$

where $d(COV(t), \eth(c(t)))$ is the distance between the center of view $COV(t)$ and cell $\eth(c(t))$, to which the agent moves from its current location $c(t)$ when it executes action $\eth$. Note that in contrast to the next location $c(t+1)$, which is one of the neighboring cells of the current agent location $c(t)$, the center of view $COV(t)$ is a cell that is chosen from among all n cells of the domain.

Finally, in the third algorithm, the next action is chosen as

$$\eth(t+1) = \underset{\eth \in \mathbb{A}}{\operatorname{argmin}}\, d(COG(t), \eth(c(t))), \tag{31}$$

where $COG(t)$ stands for the "center of gravity", which is the first moment of the probability map $P(t)$, and the remaining terms have the same meanings as above.

The Q-max and SPL algorithms used the same neural network as above and were initialized with the same parameters. As above, for all the algorithms, the agent started in the initial position $c(0) = (9,4)$ and moved over the domain with the aim of detecting $\xi = 2$ targets.

The first simulations addressed the detection of static targets, which, as above, were located at points $(5,0)$ and $(0,9)$.

The results of the detection by different algorithms are summarized in Table 1. The results represent the averages over 30 trials for each algorithm.

Table 1. Number of agent actions and the discounted cumulative information gain in detecting two static targets for the false alarm rate $\alpha = 0.5$.

Detection Algorithm	Number of Actions Up to Detection of the First Target	Number of Actions Up to Detection of the Second Target	Discounted Cumulative Information Gain
Random	25	45	13.4
EIG	17	27	17.1
COV	17	24	17.5
COG	18	29	16.1
Q-max	15	21	20.5
SPL	14	21	20.1

The table shows that the proposed Q-max and SPL algorithms outperform previously developed methods in terms of both the number of agent actions and the value of the discounted cumulative information gain.

The results of the simulations over time are shown in Figure 8. Figure 8a shows the discounted cumulative reward for the Q-max algorithm in comparison with the COV algorithm (the best heuristic algorithm) and the random detection process, and Figure 8b shows similar graphs for the SPL algorithm compared to the COV algorithm and the random detection process.

The detection by the suggested algorithms is better than the detection by the COV algorithm. Namely, the Q-max algorithm results in 20.5 units of discounted cumulative reward, while the COV algorithm obtains 17.5 units of discounted reward in the same number of steps. In other words, the Q-max algorithm is nearly 1.15 times more effective than the COV algorithm. Similarly, while the COV algorithm requires 25 steps to detect the targets, the SPL algorithm requires only 20 steps, which means that the SPL algorithm is 25% better than the COV algorithm.

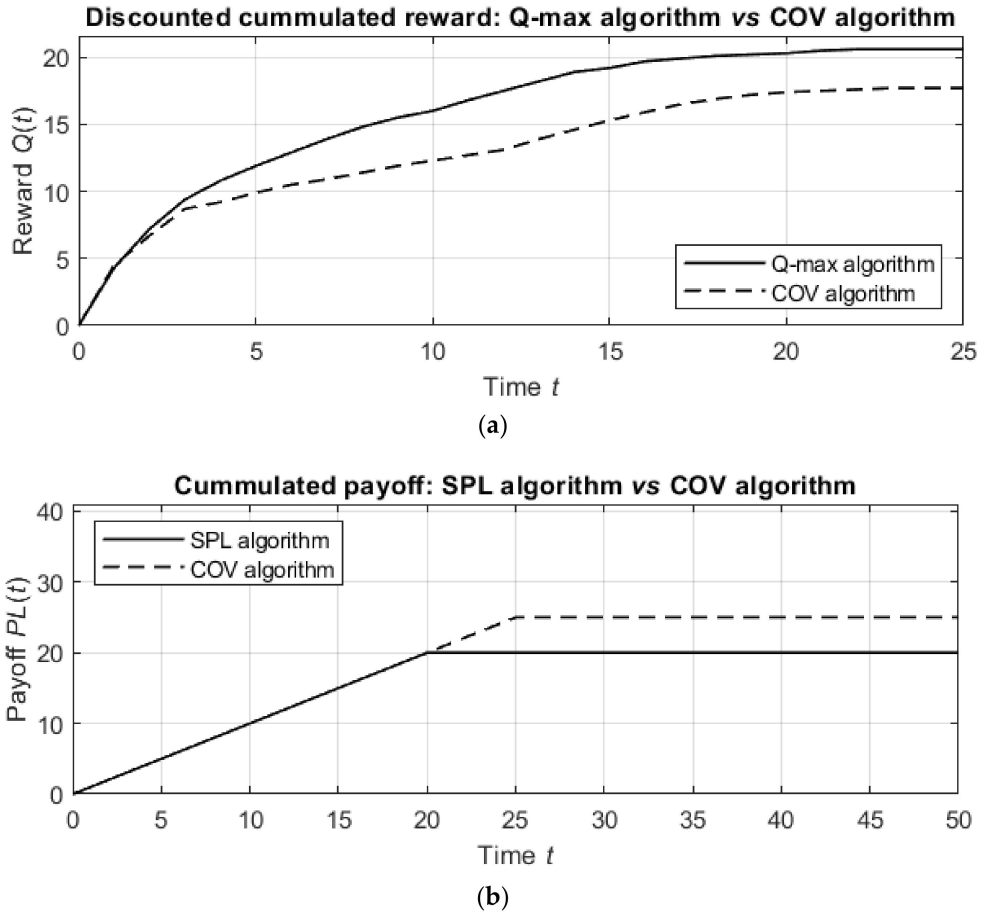


Figure 8. Cumulative reward of detection by the Q-max algorithm for static targets (**a**) and cumulative payoff of detection by the SPL algorithm for static targets (**b**) compared with the results obtained by the COV algorithm.

The second simulations addressed the detection of moving targets, which started in the initial positions $(5,0)$. and $(0,9)$. Regarding the targets' motion, it is assumed that both of them, at each time $t = 1,2,\ldots$, can apply one of the possible actions from the set $\mathbb{A} = \{\uparrow, \nearrow, \rightarrow, \searrow, \downarrow, \swarrow, \leftarrow, \nwarrow, \odot\}$ so that the probability of the action $\odot$ is $Pr\{\eth(t) = \odot\} = 0.9$ and the probability of each other action $\eth \in \mathbb{A}\backslash\odot$ is $(1-0.9)/8 = 0.0125$.

The results of detection by different algorithms (averaged over 30 trials for each algorithm) are summarized in Table 2.

Table 2. Number of agent actions and the discounted cumulative information gain in detecting two moving targets for the false alarm rate $\alpha = 0.5$.

Detection Algorithm	Number of Actions Up to Detection of the First Target	Number of Actions Up to Detection of the Second Target	Discounted Cumulative Information Gain
Random	72	105	21.8
EIG	50	65	27.1
COV	49	62	28.7
COG	55	67	26.2
Q-max	32	45	33.2
SPL	31	43	32.1

In the detection of moving targets, the suggested Q-max and SPL algorithms also outperform previously developed methods in terms of both the number of agent actions and the value of the discounted cumulative information gain.

Note that the simulation was conducted for targets with a clear motion pattern, where the probabilities of the targets' actions represent slow random motion of the targets near their initial locations. Another possible reasonable motion pattern is motion with a strong drift in a certain direction, which results in a similar ratio between the numbers of actions and the discounted cumulative information gains to that presented in Table 2.

In contrast, if the random motion of the targets is a random walk with equal probabilities $Pr\{\eth(t) = \eth\} = 1/9$ for all actions $\eth \in \mathbb{A}$, then the training becomes meaningless since both with and without training, the agent needs to detect randomly moving targets.

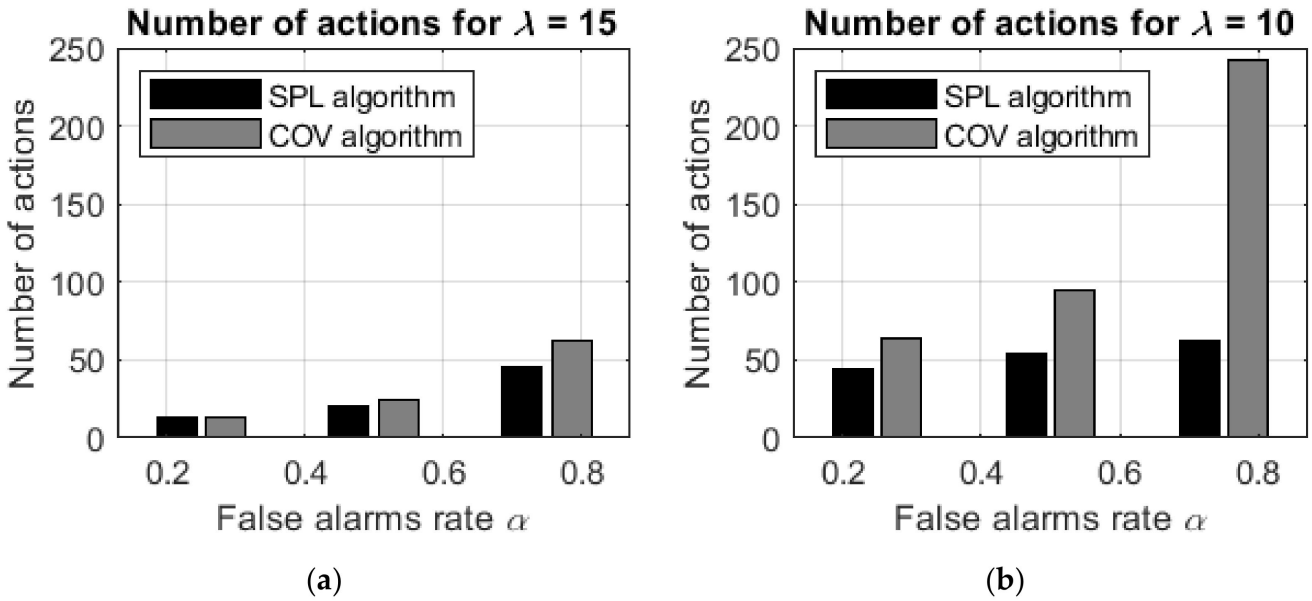
The other results obtained for the Q-max/SPL algorithms also indicated better performance by these algorithms compared with that of the heuristic algorithms. The algorithms were compared with the best heuristic COV algorithm. The results of the trials for different values of the false alarm rate α and of the sensor sensitivity λ are summarized in Table 3.

Table 3. The number of agent actions in detecting two static targets for different values of the false alarm rate α and of the sensor sensitivity λ.

Sensor Sensitivity	Algorithm	False Alarm Rate		
		α=0.25	α=0.5	α=0.75
$\lambda = 15$	COV	14	25	63
	SPL/Q-max (average)	13	20	45
$\lambda = 5$	COV	64	95	242
	SPL/Q-max (average)	44	54	63

For all values of the false alarm rate and the sensor sensitivity, the Q-max and SPL algorithms strongly outperform the best heuristic COV algorithm.

To emphasize the difference in the detection time between the suggested SPL and Q-max algorithms and the heuristic COV algorithm, the data shown in the table are depicted in Figure 9.



(**a**) (**b**)

Figure 9. The number of agent actions in detecting two static targets with the SPL/Q-max algorithms (black bars) and the COV algorithm (gray bars): (**a**) $\lambda = 15$ and (**b**) $\lambda = 10$.

As expected, the Q-max and SPL learning algorithms demonstrate better performance than the heuristic COV algorithms without learning, and the difference between the algorithms increases as the false alarm rate α increases and the sensor sensitivity λ decreases. For example, if $\lambda = 15$ and $\alpha = 0.25$, then the improvement in the number of actions is 10%, while if $\lambda = 5$ and $\alpha = 0.75$, then the improvement is significantly stronger at 75%.

In other words, computationally inexpensive heuristic algorithms provide effective results in searches with accurate sensors and a low rate of false alarms. However, in searches with less precise sensors or with a high rate of false-positive errors, the heuristic algorithms are less effective, and the Q-max and SPL learning algorithms should be applied.

4.4. Comparison between the SPL Algorithm and an Algorithm Providing the Optimal Solution

The suggested approach was compared with the known dynamic programming techniques implemented in search algorithms for moving targets [11,20]. Since the known algorithms directly address the optimal trajectory of the agent and result in an optimal path, in the simulation, we considered the SPL algorithm, which uses the same criteria as the known algorithms.

The comparisons were conducted as follows. The algorithms were trialed over the same domain with a definite number n of cells, and the goal was to reach the maximal probability $P^* = 0.95$ of detecting the target. When this probability was reached, the trial was terminated, and the number of agent actions was recorded.

Since the known algorithms [11,20] implement dynamic programming optimization over possible agent trajectories, their computational complexity is high, and for the considered task, it is $\mathcal{O}\left(n \cdot 9^t\right)$, where n is the number of cells and t is the number of actions.

Therefore, to finish the simulations in reasonable time (120 min for each trial), the algorithms were trialed on a very small case with $n = 10 \times 10 = 100$ cells. Note that in the original simulations, these algorithms were trialed on smaller cases. If the desired probability $P^* = 0.95$ of detecting the targets was not reached in 120 min, the algorithms were terminated.

In all trials, the known dynamic programming algorithms planned $t = 7$ agent actions in 120 min, while the suggested SPL algorithm, in the same time of 120 min, planned significantly more actions and reached at least the desired probability $P^* = 0.95$ of detecting the targets. The results of the comparison between the SPL algorithm and the known dynamic programming algorithms that provide optimal solutions are summarized in Table 4.

Table 4. Number of planned agent actions in detecting two static targets by the SPL algorithm and dynamic programming (DP) algorithm for different values of the false alarm rate $\boldsymbol{\alpha}$ and of the sensor sensitivity $\boldsymbol{\lambda}$.

Sensor Sensitivity	Algorithm	Characteristic	False Alarm Rate				
			α=0	α=0.05	α=0.1	α=0.25	α=0.5
$\lambda = 15$	DP	Run time	0.4 s	1 min	120 min	120 min	120 min
		Number of planned actions	3	5	7	7	7
		Detection probabilities p_1 and p_2	1.0 1.0	1.0 0.99	0.99 0.96	0.90 0.84	0.84 0.68
	SPL	Run time	0.4 s	1 min	120 min	120 min	120 min
		Number of planned actions	3	5	7	13	20
		Detection probabilities p_1 and p_2	1.0 1.0	1.0 0.99	0.99 0.96	0.99 0.95	0.99 0.95

Table 4. *Cont.*

Sensor Sensitivity	Algorithm	Characteristic	False Alarm Rate				
			α=0	α=0.05	α=0.1	α=0.25	α=0.5
$\lambda = 10$	DP	Run time	1 min	120 min	120 min	120 min	120 min
		Number of planned actions	5	7	7	7	7
		Detection probabilities p_1 and p_2	1.0 1.0	0.96 0.95	0.90 0.85	0.85 0.65	0.71 0.43
	SPL	Run time	1 min	120 min	120 min	120 min	120 min
		Number of planned actions	5	7	15	21	32
		Detection probabilities p_1 and p_2	1.0 1.0	0.96 0.95	0.97 0.95	0.98 0.95	0.99 0.95

Until termination at 120 min, the SPL algorithm plans more agent actions and results in higher detection probabilities than the DP algorithm for both values of sensor sensitivity λ and for all values of the false alarm rate α. For example, the dependence of the detection probabilities on the run time for sensor sensitivity $\lambda = 15$ and false alarm rate $\alpha = 0.25$ is depicted in Figure 10.

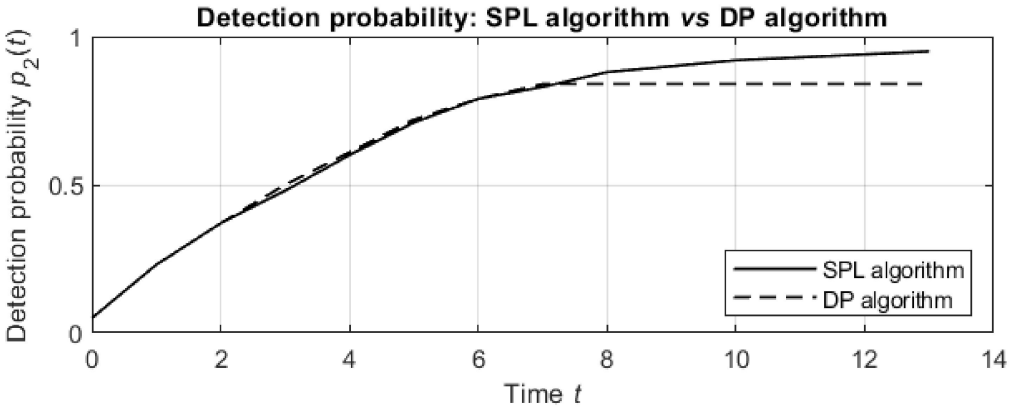


Figure 10. Dependence of the detection probabilities on the number of planned actions for the SPL algorithm (solid line) and DP algorithm (dotted line); the sensor sensitivity is $\lambda = 15$, the false alarm rate is $\alpha = 0.25$, and the termination time is $t = 120$ min.

For the first 7 actions, the detection probabilities of both algorithms increase similarly. Then, the DP algorithm does not plan additional actions in 120 min, while the SPL algorithm results in more planned actions, and the detection probabilities for these actions continue increasing until termination after 13 planned actions.

Finally, the dependence of the detection probabilities on the false alarm rate α at termination after 120 min is depicted in Figure 11.

For a low false alarm rate α, the SPL algorithm results in the same detection probabilities as the optimal DP algorithms, but for a higher false alarm rate α, the detection probabilities obtained by the DP algorithms significantly decrease (to 0.68 and 0.43 for $\lambda = 15$ and $\lambda = 10$, respectively), while the probability obtained by the SPL algorithm is 0.95 for any false alarm rate and both sensor sensitivities.

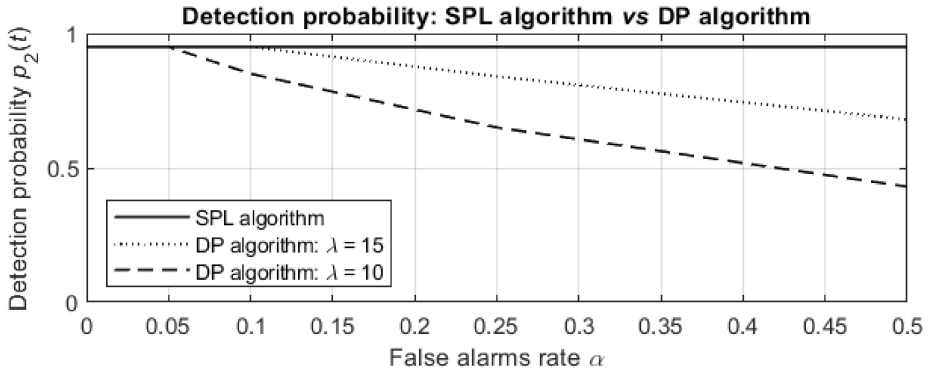


Figure 11. Dependence of the detection probabilities on the false alarm rate α for sensor sensitivities $\lambda = 15$ (dotted line) and $\lambda = 10$ (dashed line). The probability 0.95 for the SPL algorithm and all values of α is depicted by the solid line. The termination time is 120 min.

4.5. Run Times and Mean Squared Error for Different Sizes of Data Sets

Finally, we considered the dependence of the run time and mean squared error on the size of the data set. The results of these simulations are summarized in Table 5.

Table 5. Run times and temporal difference errors with respect to the size of the data set.

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 *
10×10	42,009	5000	4	0.13
		10,000	8	0.12
20×20	648,009	5000	7	0.15
		10,000	14	0.13
40×40	10,272,009	5000	10	0.18
		10,000	20	0.15

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

While the size of the domain and the number of links in the network exponentially increase, the mean squared error increases very slowly and remains small. In addition, it is seen that with an exponentially increasing domain size, the run time increases linearly, and the computations require a reasonable time even on the previously described PC. However, for realistic engineering and industrial problems with larger domains, it is reasonable to use computation systems with greater GPU power.

5. Discussion

This paper presents a novel algorithm for the navigation of mobile agents detecting static and moving hidden targets in the presence of false-positive and false-negative errors. The suggested algorithm continues in the direction of previously developed procedures [12,13] for seeking and detecting hidden targets. However, in contrast to these procedures, which follow an immediate one-step decision making process, the proposed method implements the deep Q-learning approach and neural network techniques.

The suggested algorithm is implemented in two versions: a procedure that maximizes the cumulative discounted expected information gain over the domain (Q-max algorithm) and a procedure that minimizes the expected path length of the agent in detecting all the targets (SPL algorithm). Formally, the first procedure is an extension of previously developed techniques based on the expected information gain calculated over the local neighborhood of the agent, while the second is a direct application of Q-learning techniques to the required value of the agent's path length.

The simulations show that after offline training of the neural network using the generated data set, the algorithm provides solutions that outperform the results obtained

by the previously developed procedures, both in terms of the cumulative information gain and in terms of the agent's path length. Moreover, the expected number of actions obtained by the Q-max algorithm by maximizing the cumulative discounted expected information gain is statistically equal to the number of actions obtained by the SPL algorithm by minimizing the expected path length. This equivalence follows directly from the nature of the problem: in terms of information, the detection of the targets means accumulating as much information as possible about the targets' locations, and in terms of the path length, the detection of the targets means making as few movements as possible in order to specify the exact target locations.

6. Conclusions

This paper considered the detection problem for multiple static and moving targets hidden in a domain, directly extending the classical Koopman search problem. Following previously developed methods, we addressed detection with both false-positive and false-negative detection errors.

In the exploration stage, the suggested algorithm implements the deep Q-learning approach and applies neural network techniques for learning the probabilities of the targets' locations and their motion patterns; then, in the exploitation stage, it chooses actions based on the decisions made by the trained neural network.

The research suggested two possible procedures. In the first, called the model-free procedure, the agent detects the targets in the environment and simultaneously, online, learns the environment and trains a neural network that supports the agent's decision-making processes. In the second procedure, called the model-based procedure, the agent begins detection only after offline learning and requires an exact model of the targets' activity.

The results obtained by maximizing the discounted cumulative expected information gain and by minimizing the expected length of the agent's path demonstrate that the suggested algorithm outperforms previously developed information-based procedures and provides a nearly optimal solution even in cases in which the existing techniques require an unreasonable computation time.

The proposed algorithms were implemented in the Python programming language and can be used both for further development of the methods of probabilistic search and detection and for practical applications in the appropriate fields.

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 was partially supported by the Koret Foundation Grant for Smart Cities Digital Living 2030.

Institutional Review Board Statement: Not applicable.

Informed Consent 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. Washburn, A.R. *Search and Detection*; ORSA Books: Arlington, VA, USA, 1989.
3. Koopman, B.O. *Search, and Screening. Operation Evaluation Research Group Report, 56*; Center for Naval Analysis: Rosslyn, VI, USA, 1946.
4. Stone, L.D. *Theory of Optimal Search*; Academic Press: New York, NY, USA, 1975.
5. Cooper, D.; Frost, J.; Quincy, R. *Compatibility of Land SAR Procedures with Search Theory*; US Department of Homeland Security: Washington, DC, USA, 2003.
6. Frost, J.R.; Stone, L.D. *Review of Search Theory: Advances and Applications to Search and Rescue Decision Support*; US Coast Guard Research and Development Center: Groton, MA, USA, 2001.
7. Kagan, E.; Ben-Gal, I. *Probabilistic Search for Tracking Targets*; Wiley & Sons: Chichester, UK, 2013.

8. Stone, L.D.; Barlow, C.A.; Corwin, T.L. *Bayesian Multiple Target Tracking*; Artech House Inc.: Boston, MA, USA, 1999.
9. Kagan, E.; Ben-Gal, I. *Search, and Foraging: Individual Motion and Swarm Dynamics*; CRC/Taylor & Francis: Boca Raton, FL, USA, 2015.
10. Kagan, E.; Shvalb, N.; Ben-Gal, I. (Eds.) *Autonomous Mobile Robots and Multi-Robot Systems: Motion-Planning, Communication, and Swarming*; Wiley & Sons: Chichester, UK, 2019.
11. Brown, S. Optimal search for a moving target in discrete time and space. *Oper. Res.* **1980**, *28*, 1275–1289. [CrossRef]
12. Matzliach, B.; Ben-Gal, I.; Kagan, E. Sensor fusion and decision-making in the cooperative search by mobile robots. In Proceedings of the International Conference Agents and Artificial Intelligence ICAART'20, Valletta, Malta, 22–24 February 2020; pp. 119–126.
13. Matzliach, B.; Ben-Gal, I.; Kagan, E. Cooperative detection of multiple targets by the group of mobile agents. *Entropy* **2020**, *22*, 512. [CrossRef] [PubMed]
14. Elfes, A. Sonar-based real-world mapping, and navigation. *IEEE J. Robot. Autom.* **1987**, *3*, 249–265. [CrossRef]
15. Elfes, A. Occupancy grids: A stochastic spatial representation for active robot perception. In Proceedings of the 6th Conference on Uncertainty in Artificial Intelligence, New York, NY, USA, 27–29 July 1990; pp. 136–146.
16. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
17. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; Bradford Book, MIT Press: Cambridge, MA, USA, 1998.
18. Jeong, H.; Hassani, H.; Morari, M.; Lee, D.D.; Pappas, G.J. Learning to Track Dynamic Targets in Partially Known Environments. 2020. Available online: https://arxiv.org/abs/2006.10190 (accessed on 20 June 2022).
19. 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]
20. Washburn, A.R. Search for a moving target: The FAB algorithm. *Oper. Res.* **1983**, *31*, 739–751. [CrossRef]