Development of an algorithm for managing a multi-robot system for cargo transportation based on reinforcement learning in a virtual environment

L A Rybak¹*, L Behera², M A Averbukh¹ and A V Sapryka¹

¹Belgorod State Technological University named after V. G. Shukhov, 46, Kostyukova Str., Belgorod, 308012, Russia
²Indian Institute of Technology Kanpur, Kanpur, 208016, India

E-mail: rl_bgtu@intbel.ru

Abstract. This paper is devoted to the study of varieties of the Q-Learning algorithm – deep Q-networks and dueling Q-networks. These algorithms belong to the group of reinforcement learning algorithms. Neural network architectures are selected. The process of modeling the robot’s operation in the problem of cargo delivery from a random point A to the green zone is described. The method of obtaining information about the environment by the robot using the Raycast method is described. A block diagram for controlling the robot movement has been developed, which consists of a block of positioning and state sensors, a neural network module, and a block for constructing a trajectory. The last two blocks together form a system for automatically controlling the agent’s movement in the external environment. Modeling was performed in the Unity development environment. To work with ml agents, the special Unity ML-Agents tool is used. This tool is implemented using a modern DRL servo motor, which is based on the model of the optimization algorithm proximal policy optimization (PPO). A constructive simplification of the agent and environment to facilitate the reproduction of training scenes is implemented. An algorithm for training a robot in a random environment is presented. The optimal parameters of the algorithms under consideration are selected. Suggestions are made to improve the performance of algorithms for this problem.

1. Introduction
The problem of controlling robots in a previously unknown environment was solved at different times by different methods [1-7]. Most of the work in this area is devoted to path planning, where the environment was analyzed in order to find the most convenient path. The work of the foreign scientist S.V. Wilson [8] is considered one of the most significant works in this field and led to the birth of a whole class of problems devoted to animates (ANIMAT = ANIMAL + ROBOT), that is, robots trained using a reinforcement algorithm. The classic Wilson Animat works in a discrete world and is constantly trained in the same environment. The goal of the Animat is to learn how to reach the goal from any starting position in the minimum number of steps. There are works devoted to managing an autonomous agent in a continuous environment, where the agent learns to avoid obstacles, but does not achieve any goal [9] and a solution to the control problem is proposed, but only in a 2-dimensional environment [10]. All these works are characterized by the fact that they are constantly taught in the
same environment, so it is only guaranteed that the robot can function effectively only in this environment and it is not known how effective its behavior will be if it changes slightly.

The paper will use an algorithm called time difference learning (TD, temporal difference), which belongs to the categories of learning algorithms without a model – without requiring information about the model, it can be applied to non-episodic tasks. Experiments will be presented with a robot that is trained in many typical environments, so it can function effectively in a completely unknown environment.

2. Materials and methods

As a platform for modeling, the Unity development environment was chosen with the special Unity ML-Agents tool connected – this is a new plugin in the Unity game engine that allows using Unity tools as the basis for creating an environment when training ml-agents. The robot has a task to reach the goal, avoiding collisions with obstacles. The environment is a three-dimensional space bounded around by walls. To facilitate the reproduction of scenes, all elements of the environment – agent, cargo, obstacles – are simplified and presented as geometric primitives. The agent (blue cube) appears in the center zone every time the scene is replayed. A cargo (a white parallelepiped) appears on the stage at a random location. The agent’s task is to detect the cargo, approach it, and deliver it to the green zone in the fastest possible time (figure 1).

![Figure 1. The agent (blue cube) and the load (white box) during training.](image)

The robot receives information about the environment using a special Raycast method. A Raycast is a ray emitted from some object in some direction of some length (or infinite) to detect collisions with objects. After the ray is emitted, we get the object (or an array of objects, if we use Physics.RaycastAll) that it encountered, and then we can determine whether we hit the object we need. The operation diagram of the autonomous agent sensors is shown in figure 2 a, b.
3. Results

3.1. Algorithm description

We will use the Q-learning algorithm. In Q-learning, an agent interacts with the environment iteratively by performing an action. The environment responds by informing the agent of the reward for this action and moving to the next state. This happens continuously until the environment is “resolved”. Enhanced learning tries to learn the best sequence of actions. This is done by trying different combinations of actions, at first randomly, than using a policy based on what the model has learned from rewards up to this point. This happens until the environment reaches its final state [11, 12].

We also consider two modifications of this algorithm – Deep Q-network (DQN) and Duel Deep Q-network. As the standard Q-learning method is best suited for environments with a finite number of states and limited sets of actions, in which an exhaustive search for the optimal Q value was possible for all possible action/state pairs [10]. But most often they need to create an environment in which the number of states is very large, and in each state there are many actions available. Going through all the actions in each state would take too long. Another, more efficient approach is based on the approximation of a Q-function with some parameter $\theta$ (1).

$$Q(s,a;\theta) \approx Q'(s,a)$$  \hspace{1cm} (1)

They need to use a neural network with weights to approximate the value of Q for all possible actions in each state, it can be called a Q-network. However, the problem of network training and the problem of the type of objective function appear. The update rule for Q-learning looks like this:

$$Q(s,a) = Q(s,a) + \alpha (r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$ \hspace{1cm} (2)

where $r + \gamma \max_{a'} Q(s',a')$ – target value, and $Q(s,a)$ – predicted value. We strive to minimize this value by using the correct policy.

Similarly, in a Deep Q-network or DQN, they can define the loss function as the square of the difference between the target and predicted values. We also try to minimize losses by updating the weights $\theta$.

$$E = (y_i - Q(s,a;\theta))^2$$ \hspace{1cm} (3)

where $y_i = r + \gamma \max_{a'} Q(s',a';\theta)$.

Gradient descent is used to update the weights and minimize losses. Therefore, in DQN neural networks are used as approximation functions for the Q-function, and gradient descent is used to minimize errors (figure 3).
Figure 3. Deep q network architecture.

The Q function determines how good it is for the agent to perform action $a$ in state $s$, and the value function determines how good it is for the agent to stay in state $s$. Let us introduce a new function called the advantage function. It can be defined as the difference between the value function and the average value of this state. The advantage function indicates how well the agent performs this action compared to other actions.

Thus, the value function sets the desirability of a state, and the advantage function sets the desirability of an action. When combining these functions, the result will tell us how desirable it is for the agent to perform action $a$ in state $s$, that is, in fact, we get a Q-function, which can be defined as the sum of the value function and the advantage function:

\[ Q(s, a) = V(s) + A(s) \] (4)

Figure 4 shows the dueling network architecture DQN:

Figure 4. Dueling DQN architecture.

The dueling DQN architecture is almost identical to the DQN architecture, except that the fully connected layer is divided into two threads at the end. One thread calculates the value function, and the other calculates the advantage function. At the end, these two streams are combined by an aggregating layer, resulting in a Q-function [13].

3.2. Synthesis of the learning algorithm

The training process is divided into stages (tests). At each stage of training, the robot and the target are randomly placed at a new point in space. A new location for all obstacles is also randomly created. The reward that the robot received only at the end of the stage, in all other cases, the reward was zero. The stage ended when the robot reached the goal, when it encountered an obstacle, or because of a timeout. Different reward values are defined for all these three cases:

1) if the robot reaches the goal, it gets +1 to the reward;
2) when the robot collides with an obstacle, the reward value is calculated using the following formula:
\[ r = 0.5 \cdot \exp(-2d_{\text{goal}} / l_{\text{room}}) \]  \hspace{1cm} (5)

where \( d_{\text{goal}} \) – distance to the goal, and \( l_{\text{room}} \) – length of one side of the environment;

3) at inaction the robot receives a reward that is equal to the collision reward +0.3.

Four randomly generated convex quadrilaterals were used as obstacles in the learning process. At each stage of operation, the robot can choose one of five possible actions: move forward, move forward at an angle of 15 degrees to the left, move forward at an angle of 15 degrees to the right, turn 15 degrees to the left, and turn 15 degrees to the right. The robot motion control scheme is shown in figure 5.

At each iteration of the robot appears in the generated environment, where it performs some actions based on the readings of the positioning and status sensor block. All data from this block is sent to the block of the neural network module and to the block of building the trajectory of movement, which gives commands to the robot to improve actions to achieve results. The robot agent performs actions in the external environment. In turn, the external environment transmits data about the agent’s state and the result of its actions to the neural network module block.

3.3. Software implementation

The program uses several important hyperparameters that affect learning. The number of episodes is 2000, the discount factor is 0.99, the training series size is 64, and the number of hidden network layers is 256. Full characteristics of hyperparameters are shown in table 1.

| Hyperparameter                  | Value       |
|--------------------------------|-------------|
| Number of Episodes             | 2000        |
| Number of Timesteps            | 1000        |
| Print Checkpoint step every    | 4           |
| Training Batch Size            | 64          |
| Discount Rate / Gamma          | 0.99        |
| Learning Rate / alpha          | 5.00E−04    |
| Number of Hidden Layers        | 256         |
| Fully Connected Layer 1 Units  | 64          |
| Fully Connected Layer 2 Units  | 64          |
| TAU                            | 1.00E−03    |
| Epsilon                        | 0.1         |
| Epsilon–Min                    | 0.01        |
| Epsilon–Decay                  | 0.995       |
Figure 6 shows the working window of the simulator while training the agent to deliver cargo from a random point to the green zone without obstacles (purple parallelepipeds).

![Figure 6](image)

**Figure 6.** The process of training an agent in an environment with obstacles.

Figure 7 a, b shows the evaluation results obtained during model training. The two approaches had very different training behaviors. The Vanilla method showed a better start to the game score, with a faster increase in score over the first 250 episodes. However, after 500 episodes, both models converge on a similar pattern.

![Figure 7a](image) ![Figure 7b](image)

**Figure 7a.** Training quality indicators for a standard deep q-network.  
**Figure 7b.** Quality indicators of the dueling deep q-network.

Using the Vanilla method, a score of +15 was achieved in 260 episodes, while Dueling Networks achieved these score in 360 episodes.

4. Discussion
All the research discussed above is aimed at developing and testing a management system based on reinforcement learning. The analysis of reinforcement learning algorithms, namely the Q-learning algorithm and its modifications – Deep Q-learning and Duel Q-learning. Simulation of learning and experience reproduction by a robot in the Unity development environment with the ML-Agents module is performed. A block diagram of the developed system is constructed. The choice of hyperparameters of the studied neural network is made. The performance indicators of the standard deep Q-network and the dueling network are analyzed. Dueling networks were expected to improve on the standard method.
5. Summary
The resulting improvements can be explained by the fact that this model was not configured with any hyperparameter setting. However, regardless of the basic values of the hyperparameter, the agent is able to quickly and efficiently solve the environment.

6. References
[1] Rybak L A, Khalapyan S Y, Malyshev D I 2019 Optimization of the trajectory of movement of a parallel robot based on a modified ACO algorithm *IOP Conference Series: Materials Science and Engineering* **560**(1) 012036
[2] Khalapyan S Y, Rybak L A, Malyshev D I 2020 Two-Stage Method for Controlling the Movement of a Parallel Robot Based on a Planar Three-Revolute-Prismatic-Revolute Mechanism *Advances in Intelligent Systems and Computing* **1126** 368-379.
[3] Ballesteros-Escamilla M, Chairez I, Boltianski V G, Poznyak A 2018 Realization of robust optimal control by dynamic neural-programming *IFAC-PapersOnLine* **51**(13) 468-473
[4] Rybak L A, Mamaev Y A, Virabyan L G 2016 Correction algorithms synthesis for the motion path of the hexapod robot output link based on the theory of artificial neural networks *Bulletin of BSTU named after V G Shukhov* **12** 142-151
[5] Reddy T K, Arora V, Behera L. 2020 HJB-Equation-Based Optimal Learning Scheme for Neural Networks with Applications in Brain-Computer Interface *IEEE Transactions on Emerging Topics in Computational Intelligence* **4**(2) 159-170
[6] Eremenko Y I, Khalapyan S Y, Anpilov A O 2020 Improving energy efficiency of iron ore concentrate dehydration through automation using neural technologies *Gornyi Zhurnal* **3** 62-66
[7] Ballesteros M, Chairez I, Poznyak A 2020 Robust optimal feedback control design for uncertain systems based on artificial neural network approximation of the Bellman’s value function *Neurocomputing* **413** 134-144
[8] Wilson S W 1985 Knowledge growth in an artificial animal *Proceedings of the First International Conference on Genetic Algorithms and their Applications* 16-23.
[9] Prescott T J, Mayhew J E W 1992 Obstacles avoidance through reinforcement learning *Advances in Neural Information Processing system* **4** 523-530.
[10] Kuzmin V 2003 Using neural networks in the q-learning algorithm *Transport and Telecommunication* **4**(1) 74-86.
[11] Testing the Q – learning algorithm on popular game platforms [Electronic resource]. URL: https://habr.com/ru/post/345656/ (date of access 05/09/2020).
[12] Yogi S C, Tripathi V K, Kamath A K, Behera L 2019 Q-learning Based Navigation of a Quadrotor using Non-singular Terminal Sliding Mode Control *IEEE International Conference on Robot and Human Interactive Communication* **8956365**
[13] Bharat R, Bosag Zadeh R 2019 TensorFlow for Deep Learning *BHV-Petersburg* 256

Acknowledgments
The research was funded by RFBR within the research project No 18-57-45014, using equipment of High Technology Center at BSTU named after V G Shukhov.