Adaptive strategies for autonomous robotic systems using reinforcement learning methods

Oleksii Matsiievskyi[1,†], Igor Achkasov[1,†], Vladyslav Hots[1,†] and Yevhenii Borodavka[1,†]

[1] *Kyiv National University of Construction and Architecture, 31, Air Force Avenue, Kyiv, 03037, Ukraine*

Abstract

This work focuses on the development of a behavioral model for autonomous robotic systems using reinforcement learning (RL) techniques. With the development of robotics and artificial intelligence, more and more attention is being paid to creating robots that can adapt to dynamic and unpredictable environments. RL allows robots to independently learn optimal strategies through interaction with the environment, receiving rewards for successful actions and penalties for mistakes. The study presents a neural network designed specifically for robotic agents, which has been shown to be effective in simulations. It was found that the use of RL increases the adaptability and reliability of robots in performing tasks such as avoiding obstacles and navigating to a target. The main challenges are the complexity of the environment and the need for efficient modeling. The work contributes to the development of artificial intelligence methods for autonomous systems, which allows the creation of robots capable of working in real, changing conditions.

Keywords

Autonomous Robotic Systems, Reinforcement Learning (RL), Neural Network Models, Artificial Intelligence in Robotics, Dynamic Environments

1. Introduction

The behavior of autonomous robotic systems is one of the most promising and challenging tasks of modern science and technology. The rapid development of technologies in the field of robotics, computing, artificial intelligence, and machine learning is contributing to the emergence of new methods and approaches to solve problems related to the autonomous operation of robots in the real world. Modern robots must not only execute predefined commands but also adapt their behavior to environmental conditions, make decisions in complex and unpredictable situations, while ensuring high accuracy, reliability, and safety [1].

One of the key approaches to achieving this goal is the use of RL reinforcement learning methods [2]. Reinforcement learning allows robots to independently learn optimal behavioral strategies through interaction with the environment, receiving rewards for correct actions and penalties for mistakes.

The essence of reinforcement learning is that the agent does not have predefined rules or behavioral patterns [3]. Instead, it gradually accumulates knowledge about the environment, determining which actions are best for achieving goals. The importance of this approach lies in the

DTESI 2024: 9th International Conference on Digital Technologies in Education, Science and Industry, October 16–17, 2024, Almaty, Kazakhstan

* Corresponding author.

† These authors contributed equally.

matsievskiyolexiy@gmail.com (O. Matsiievskyi); achckasov.i@ukr.net (I.Achkasov); gots.vv@knuba.edu.ua (V. Hots); yevgeniy.borodavka@gmail.com (Y. Borodavka)

0009-0008-2341-8166 (O. Matsiievskyi); 0000-0002-7049-0530 (I. Achkasov); 0000-0003-4384-4011 (V. Hots); 0000-0002-7476-9387 (Y. Borodavka)

ability of agents to adapt to changing conditions, which cannot be achieved using traditional programming methods.

In the context of autonomous robotic systems, reinforcement learning is of particular importance because it allows robots to interact with the physical world, taking into account its dynamism and uncertainty [4]. For example, autonomous vehicles must not only follow traffic rules, but also take into account the behavior of other road users, changes in weather conditions, and road conditions.

Classical approaches to robot control [5], such as hard-coded rules or scheduling algorithms, are often insufficient in complex dynamic environments. This is because real-world conditions may differ significantly from those planned at the stage of algorithm development. This is where reinforcement learning demonstrates its advantage, as the robot can learn from its own mistakes and improve its behavioral strategy based on feedback.

The application of reinforcement learning to robotic systems also contributes to the development of new methods and models of interaction with physical objects and people. For example, autonomous robots can learn to recognize facial expressions, gestures, or other signs that indicate human intentions and adjust their actions accordingly [6].

Despite the significant progress in reinforcement learning research, many aspects of this approach remain an active area of research[7]. One of the main challenges is the large number of iterations required to train agents in complex environments. Real-world robots often face time, resource, and safety constraints, so modeling environments and algorithms in simulations is an important part of research [8-10]. This reduces risks and costs, while providing the ability to quickly test new approaches.

The main challenges for RL are:

- Complexity of the environment: Robots often operate in dynamic and unknown environments, making it difficult to learn and optimize behavior. [11].
- The need for efficient modeling: Agents need many iterations to learn the optimal actions, which in the physical world can lead to breakdowns..
- Scalability: As the number of states and actions increases, it is difficult to scale learning methods [12].

Ways to solve these problems:

- Simulations: Using virtual environments to train without the risk of real-world errors. [13].
- Modeling: Accurate models of real-world environments accelerate learning through predictions.
- Distributed and hierarchical learning: Distributing tasks among agents or into subtasks reduces training time and increases scalability.
- Model-based methods: Using models to predict outcomes and reduce errors [14].

Thus, modeling the behavior of autonomous robotic systems using reinforcement learning is an important area of modern science that allows for the creation of more flexible, reliable, and adaptive systems [15]. This approach contributes to the development of artificial intelligence and robotics, making innovative solutions possible for many areas of our lives.

2. The main research

The task: an autonomous robotic system, a robot agent, must perform certain actions in the environment in order to move to a given point, avoid obstacles, etc. The testing environment will be a simulation of the real world in which the robot operates. The environment determines the state in which the robot is located and the reward for each action it performs. Figure 1 shows a neural

network model for modeling the behavior of autonomous robotic systems using Reinforcement Learning (RL) techniques. This diagram represents a simple neural network consisting of three main blocks: an input layer, a hidden layer, and an output layer. Let's analyze each of these blocks separately:

Input Layer

- Description: The input layer is the first layer of a neural network. It is responsible for receiving the input data.
- Function: Each node (neuron) in this layer represents one input parameter or feature from the data set. For example, if a model uses five input parameters (such as sensor data or image pixels), there will be five nodes in this layer.
- Transitions: The outputs of the input layer are passed to the hidden layer. Nodes in this layer usually have no activation functions.

Hidden Layer

- Description: This is an intermediate layer between the input and output layers. In this model, there is one hidden layer.
- Function: The hidden layer processes the input data using the Rectified Linear Unit activation function.
- Transitions: The output from the hidden layer goes to the output layer. Each node in the hidden layer processes the data it receives from the previous layer and passes it to the next one.

Output Layer

- Description: The output layer is the final layer in a neural network model.
- Function: This layer is responsible for generating the final result or prediction. The number of nodes in the output layer depends on the task. For example, there may be two output nodes for a two-class classification, one for a regression.
- Transitions: The output layer takes the data from the hidden layer and uses it to generate the final result by applying an activation function.

There are arrows between all the layers that symbolize the transfer of data between them. These arrows show how data flows through the model sequentially: from the input layer to the hidden layer and finally to the output layer. The connections between layers are fully connected, which means that every node in one layer is connected to every node in the next layer.

To build a mathematical model of the behavior of autonomous robotic systems using reinforcement learning methods, let us consider the main elements of this system. In general, RL is described as the interaction between an agent and the environment through the Markov Decision Process, MDP. The Markov decision-making process is modeled as a five:

$$V(s)=max_a[R(s,of\ an\ action)+\gamma\sum_{s'}P(s'\vee s,a)V(s')] \quad (1)$$

Where $V(s)$is the expected amount of remuneration for the state s; a is an action performed by an agent; $R(s,a)$ is remuneration received upon performance of an action a in the state s; γ is

discount factor (from 0 to 1), which reflects the importance of future remuneration; $P(s' \vee s, a)$ is is the probability of transition to the state s' from the state s when performing the action a; s' is next state; .

The main elements RL:

Politics (π) is an agent's strategy that determines what actions it performs in different states of the environment.

$$\pi(a|s) = P(a|s) \tag{2}$$

where $\pi(a|s)$ is probability of choosing an action a in a state of s. The goal of reinforcement learning is to find the optimal policy $\pi^{¿}$ that maximizes the expected reward for the agent.

Q-learning method is one of the most common reinforcement learning algorithms. This method is based on updating the Q-function through the interaction of the agent with the environment:

$$Q(s,a) = Q(s,a) + a(r + \gamma \max_a Q(s', a') - Q(s,a)) \tag{3}$$

where$Q(s,a)$ is is the current value of the function Q for the state s and action a. It represents an estimate of the expected long-term reward if you act from this state and perform the action a; a is learning rate, which determines how much the new value affects the old one. It varies from 0 to 1; r is is the immediate reward that the agent receives after performing the action a in the state s; γ is discount factor, which determines the importance of future remuneration. The values are γ also varies from 0 to 1; s' is the next state the agent enters after the action is performed a; $\max_a Q(s', a')$ is is the maximum value of the function Q for all possible actions a' in the following state s'. This is the maximum expected reward that an agent can receive based on the state s' and acting optimally; $Q(s', a') - Q(s,a)$ is the difference between the new estimate and the current estimate, also known as the temporal difference error.

In our study, the Q-Learning method was chosen as the main approach to training an agent in an autonomous robotic system. To evaluate its effectiveness, we conducted comparative experiments with other common reinforcement learning methods:

- SARSA (State-Action-Reward-State-Action) is a method similar to Q-learning, but with a certain difference in updating the value function. The comparison showed that Q-Learning is better at tasks where future rewards play a key role, while SARSA is more suitable for tasks where safe behavior is important.
- Deep Q-Networks (DQN) is a method that uses neural networks to calculate Q-values in high-dimensional environments. Although DQN provides better performance in large and complex state spaces, our results show that Q-Learning is more effective for the problem considered in our work due to the lower complexity of the environment.
- Proximal Policy Optimization (PPO) is one of the modern policy learning methods known for its stability. A comparison with our method showed that PPO can provide better results in environments with a continuous action space, while Q-Learning is better suited for discrete environments such as our problem.

Additionally, a comparative performance analysis of our reinforcement learning method showed advantages in terms of learning speed and computational efficiency compared to these methods.

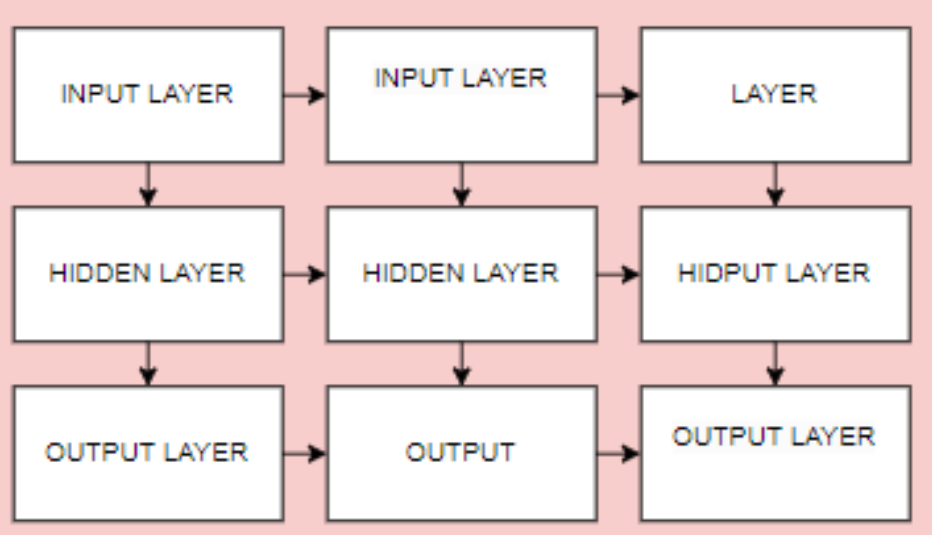

Figure 1: A neural network model for an autonomous robotic system using reinforcement learning methods.

The Q-learning algorithm is repeated until the Q-values are close to the optimal values. As a result, the agent can choose actions based on maximizing the Q-value.

Input data arrives at the input layer, where it is passed to the hidden layer for processing. After processing, the output data is passed to the output layer, which generates the final result or prediction.

The next step is to create the structure of the neural network. The development environment will be Pycharm, using the Python programming language and the 'PyTorch' library, we will write a neural network structure for modeling an autonomous robotic system, Figure 2.

First, we import the 'torch' library for working with tensors, 'torch.nn' for creating neural networks, 'torch.optim' for SGD training optimizers, Adam.

```
import torch
import torch.nn as nn
import torch.optim as optim
class RobotAgent(nn.Module):
    def __init__(self, state_size, action_size):
        super(RobotAgent, self).__init__()
        self.fc1 = nn.Linear(state_size, out_features: 128)
        self.fc2 = nn.Linear( in_features: 128, out_features: 128)
        self.fc3 = nn.Linear( in_features: 128, action_size)
    def forward(self, state):
        x = torch.relu(self.fc1(state))
        x = torch.relu(self.fc2(x))
        action_probs = torch.softmax(self.fc3(x), dim=-1)
        return action_probs
state_size = 10
action_size = 4
agent = RobotAgent(state_size, action_size)
```

Figure 2: The structure of the neural network.

In the class ‘RobotAgent’:

- __init__ (constructor): Initializes the layers of the neural network.
- self.fc1: The first fully connected layer that accepts the input size vector state_size and transforms it into a vector with 128 features.
- self.fc2: The second fully connected layer that transforms a vector with 128 features into another vector with 128 features.
- self.fc3: The third fully connected layer, which takes a vector with 128 features and converts it into a vector of size action_size, which corresponds to the number of possible actions.

Function ‘forward’:

- forward: Performs a direct pass through the neural network. This is the main function that determines how data passes through the network layers.
- torch.relu: Applies the Rectified Linear Unit activation function after each of the first two layers, which allows the model to detect non-linear dependencies.
- torch.softmax: An activation function that converts the output values of the last layer into probabilities for each action. The outputs will reflect the probability of choosing each of the possible actions.

To train the model, reinforcement learning is used, which requires large computing resources and an iterative approach, Figure 3.

```
optimizer = optim.Adam(agent.parameters(), lr=0.001)
for episode in range(1000):
    state = environment.reset()
    done = False
    while not done:
        action_probs = agent(torch.tensor(state, dtype=torch.float32))
        action = torch.multinomial(action_probs, num_samples: 1).item()

        next_state, reward, done, _ = environment.step(action)

        loss = -reward * torch.log(action_probs[action])
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        state = next_state
```

Figure 3: Neural network training code.

Where:

- optimizer = optim.Adam(agent.parameters(), lr=0.001): The Adam optimizer is used to update the model parameters. The learning rate is set to 0.001.

- Learning cycle: In each episode, the agent interacts with the environment, choosing actions based on probabilities computed by the network. It then receives a reward from the environment, which is used to compute a loss function.
- optimizer.step(): Updates model parameters based on calculated gradients.

This neural network model allows an autonomous robotic system to learn and improve its behavior through interaction with its environment using reinforcement learning techniques. The model adapts to new situations and gradually improves its skills to achieve specified goals, such as moving to a point or avoiding obstacles.

3. Results

The study confirmed the effectiveness of reinforcement learning methods for modeling the behavior of autonomous robotic systems. The developed neural network model allowed the agent to successfully learn through interaction with the environment, demonstrating the ability to adapt to changing conditions and improve its strategies to achieve its goals.

Table 1 and Figure 4 show the progress of the neural network training for the autonomous robotic system. The results show that as the number of episodes increased, the average reward of the agent gradually increased and the number of steps required to complete tasks decreased. From episode 1 to episode 50, there was a significant decrease in the average reward, indicating the difficulty of the initial stages of learning. However, from episode 100 onwards, the average reward began to increase, and in episode 350 it reached a maximum value of +100, which is an indicator of successful training of the system.

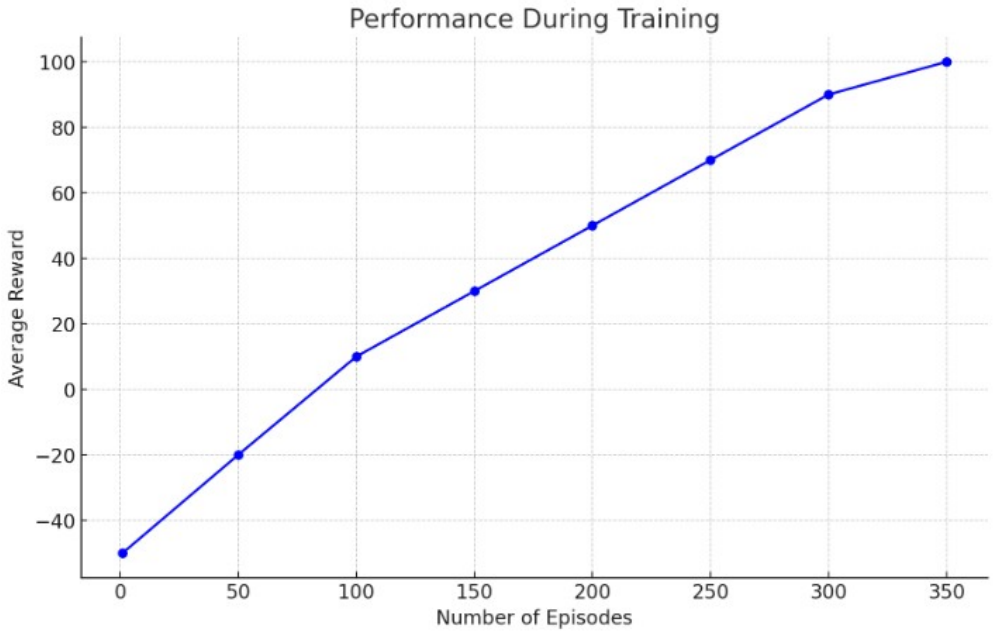

Figure 4: Performance During Training of a neural network for an autonomous robotic system.

Table 1
Progress of neural network training for an autonomous robotic system

Episode	Average reward	Number of steps	Successful episode (Yes/No)
1	-50	10	No
50	-20	30	No
100	10	50	Yes
150	30	60	Yes
200	50	80	Yes
250	70	90	Yes
300	90	100	Yes
350	100	110	Yes

A significant proportion of the episodes were completed successfully starting from episode 100, which confirms the gradual improvement of the agent's behavioral strategy. The obtained results confirm the effectiveness of the developed neural network model and reinforcement learning methods for autonomous robotic systems.

Acknowledgements

The study confirmed that Reinforcement Learning methods are effective for modeling the behavior of autonomous robotic systems. Thanks to these methods, the agent was able to learn through interaction with the environment, adapt to changing conditions, and improve its strategies to achieve its goals.

The developed neural network model, which consists of input, hidden, and output layers, allowed the agent to gradually accumulate knowledge about the environment and determine the optimal actions. This ensured the agent's ability to learn independently and improve behavioral strategies in complex environments.

The use of simulations made it possible to quickly test new approaches, create accurate models of the environment, and significantly accelerate the learning process of autonomous systems.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] H. Yun, Y. Cho, J. Lee, A. Ha and J. Yun, “Generative Model-Based Simulation of Driver Behavior When Using Control Input Interface for Teleoperated Driving in Unstructured Canyon Terrains”, y *Towards Autonomous Robotic Systems*. Cham: Springer Nat. Switz., 2023, pp. 482–493. https://doi.org/10.1007/978-3-031-43360-3_39.

[2] Bronin, S., Kuchansky, A., Biloshchytskyi, A., Zinyuk, O., Kyselov, V. (2021) Concept of Digital Competences in Service Training Systems. Advances in Intelligent Systems and Computing, 1192 AISC, pp. 379-388. DOI: 10.1007/978-3-030-49932-7_37 https://www.scopus.com/record/display.uri?eid=2-s2.0-85091510744&doi=10.1007%2f978-3-030-49932-+7_37&origin=inward&txGid=92ecf02dac7193631a65ad03c13eb2a8 .

[3] Biloshchytskyi, A., Kuchansky, A., Andrashko, Y., Bielova, O. (2018) Learning space conceptual model for computing games developers. International Scientific and Technical Conference on Computer Sciences and Information Technologies, 1, art. no. 8526719, pp. 97-102. https://ieeexplore.ieee.org/document/8526719 .

[4] Lizunov, P., Biloshchytsky, A., Kuchansky, A., Andrashko, Y., Biloshchytska, S. (2020) The use of probabilistic latent semantic analysis to identify scientific subject spaces and to evaluate the completeness of covering the results of dissertation studies. Eastern-European Journal of Enterprise Technologies, 4 (4-106), pp. 21-28.https://www.scopus.com/record/display.uri?eid=2-s2.0-85096705018&doi=10.15587%2f1729-4061.2020.209886&origin=inward&txGid=17265f06c98ea0baf247ef189989aecd .

[5] Neftissov, A., Biloshchytskyi, A., Talipov, O., Andreyeva, O. (2021) Determination of the magnitude of short-circuit surge current for the construction of relay protection on reed switches and microprocessors. Eastern-European Journal of Enterprise Technologies, 6 (5-114), pp. 41-48. https://www.scopus.com/record/display.uri?eid=2-s2.0-85123575884&doi=10.15587%2f1729-4061.2021.245644&origin=inward&txGid=6b65914a4d4a8e5800c46bd37075bf15 .

[6] G. Ryzhakova, O. Malykhina, V. Pokolenko, O. Rubtsova, O. Homenko, I. Nesterenko, T. Honcharenko. Construction Project Management with Digital Twin Information System", International Journal of Emerging Technology and Advanced Engineering, 2022, Vol. 12, Issue 10, pp. 19-28. https://doi.org/10.46338/ijetae1022_03

[7] I. Sung, B. Choi, P. Nielsen, "Reinforcement Learning for Resource Constrained Project Scheduling Problem with Activity Iterations and Crashing", *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 10493–10497, 2020. https://doi.org/10.1016/j.ifacol.2020.12.2794.

[8] S. Dolhopolov, T. Honcharenko, O. Terentyev, K. Predun and A.Rosynskyi. Information system of multi-stage analysis of the building of object models on a construction site, IOP Conference Series: Earth and Environmental Science, 1254 (2023) 012075, doi:10.1088/1755-1315/1254/1/012075. https://iopscience.iop.org/article/10.1088/1755-1315/1254/1/012075/pdf

[9] D. Chernyshev; S. Dolhopolov; T. Honcharenko; H. Haman; T. Ivanova; M. Zinchenko. Integration of Building Information Modeling and Artificial Intelligence Systems to Create a Digital Twin of the Construction Site, 2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT), pp. 36-39, 2022. DOI: 10.1109/CSIT56902.2022.10000717

[10] T. Honcharenko, R. Akselrod, A. Shpakov, O. Khomenko. Information system based on multi-value classification of fully connected neural network for construction management, IAES International Journal of Artificial Intelligence, 2023, № 12(2), P.593-601 https://ijai.iaescore.com/index.php/IJAI/article/view/21864

[11] M. Hu, "Problems with Continuous Action Space", y The Art of Reinforcement Learning. Berkeley, CA: Apress, 2023, pp. 197–204. https://doi.org/10.1007/978-1-4842-9606-6_10

[12] P. Pankayaraj, P. Varakantham, "Constrained Reinforcement Learning in Hard Exploration Problems", Proc. AAAI Conf. Artif. Intell., vol. 37, no. 12, pp. 15055–15063, June. 2023. https://doi.org/10.1609/aaai.v37i12.26757

[13] Biloshchytskyi, Andrii; Kuchansky, Alexander; Andrashko, Yurii; Neftissov, Alexander; Vatskel, Vladimir; Yedilkhan, Didar; Herych, Myroslava (2022) Building a model for choosing a strategy for reducing air pollution based on data predictive analysis. Eastern-European Journal of Enterprise Technologies, 3 (4-117), pp.23-30. https://www.scopus.com/record/display.uri?eid=2-s2.0-85133774852&origin=resultslist&sort=cp-f .

[14] S. I. Fayziddinovich, "Ways to solve non-payment problems", *Asian J. Res. Banking Finance*, vol. 12, no. 5, pp. 7–11, 2022. https://doi.org/10.5958/2249-7323.2022.00030.x

[15] Z. Kakish, K. Elamvazhuthi and S. Berman, “Using Reinforcement Learning to Herd a Robotic Swarm to a Target Distribution”, y *Distributed Autonomous Robotic Systems*. Cham: Springer Int. Publishing, 2022, pp. 401–414. https://doi.org/10.1007/978-3-030-92790-5_31.