DOI: 10.65731/ama/2026-0067

DESIGN OF NEURAL NETWORK TO GUIDE A MOBILE ROBOT TOWARDS A MOVING VIRTUAL TARGET

Cezary KOWNACKI*, Mariusz BOGDAN**

*,** Faculty of Mechanical Engineering, Department of Manufacturing Processes Automation,
Bialystok University of Technology, ul. Wiejska 45A, 15-351 Bialystok, Poland

c.kownacki@pb.edu.pl, m.bogdan@pb.edu.pl

received 10 December 2025, revised 23 July 2026, accepted 10 August 2026

Abstract: Today, research on mobile robot control and navigation is focused on plenty of analytical approaches through control loop integration of various controllers, path planning algorithms, and ending with machine learning approaches, genetic and optimization algorithms, and finally on artificial neural network applications. In the paper, there is a concept and design process of a neural network that can guide a differential-drive robot towards a virtual target that moves along different path shapes without the use of any other traditional control loop or algorithms. The dataset to train the neural network is collected by running a series of numerical simulations based on an artificial potential field approach as reference control and applying a number of various path scenarios and the tracked target's speed. As a result, a guidance artificial neural network (GNN) with a specified number of neurons in the hidden layer and a type of activation function was found. GNN can guide a robot in any case of path shape and speed, even if it was not considered in the training process. The ability of GNN to drive robot tracking of a virtual target is proven by a series of numerical simulations. The results achieved of the proposed control approach, where the GNN is the only controller applied to the robot's guidance, can be utilized to design more complex neural network structures. In such structures, outputs of another higher neural network can be used as a source of input data for the GNN, giving, i.e., the coordinates of the tracked point. This approach enables the possibility of the creation of a hierarchical bio-inspired control structure, allowing for the introduction of a term of AI-driven robot guidance, which will be the subject of future work.

Key words: mobile robot guidance, neural network, artificial potential field, virtual target tracking

1. INTRODUCTION

The control of wheeled mobile robots (WMRs) has been an area of sustained scientific interest due to its dual theoretical and practical significance. Differential-drive mobile robots (DDMRs), characterized by two independently driven wheels and a passive support caster, are particularly widespread because of their mechanical simplicity and ease of control implementation. However, achieving stable and accurate real-time motion control remains a major challenge, primarily due to nonholonomic constraints, parameter uncertainties, wheel slip, and dynamic environmental changes. Conventional control strategies, such as proportional-integral-derivative (PID) or backstepping controllers, often require precise system modeling and are limited in adapting to unknown or time-varying conditions [1÷3].

In recent years, the rise of artificial neural networks (ANNs) has significantly expanded the possibilities of intelligent control. ANNs can approximate nonlinear mappings between inputs and outputs without an explicit model, making them highly suitable for systems with uncertainties or complex dynamics [4÷6]. In mobile robotics, neural controllers are increasingly used to learn inverse kinematics or direct torque-velocity relationships, enabling adaptive behaviors that outperform fixed-parameter controllers in nonstationary environments [7, 8].

The majority of the research, however, has focused on trajectory planning and trajectory-tracking problems, where the robot is required to follow a predefined path [9÷12]. Studies such as those by Cardona and Serrano [3] and Nguyen and Le [2] proposed neural-network-based adaptive tracking controllers for nonholonomic robots, achieving robustness against disturbances and wheel slip. Similarly, Trinh et al. [6] and Juhi et al. [12] developed hybrid neural-PID and robust neural control approaches for accurate path following. While these methods demonstrate excellent tracking precision, they depend heavily on precomputed trajectories and do not address reactive, real-time guidance towards moving targets.

A smaller but significant body of work has examined real-time motion control of mobile robots using ANNs to directly generate control actions from sensory feedback. For instance, Hu and Yang [7] developed an early neural controller for a nonholonomic robot with unknown dynamics, demonstrating feasibility for online control. Liu et al. [8] later proposed an adaptive ANN motion controller compensating for wheel slip, validated experimentally on differential-drive platforms. Lopez-Franco et al. [9] integrated stereo vision with neural control for closed-loop navigation, highlighting the potential of multimodal learning in robot steering.

Recent studies have extended these ideas with deep learning and reinforcement learning approaches [13÷16]. Altawaitan et al. [17] introduced a Hamiltonian-based learning framework combining neural networks and system dynamics inference. Other contributions explored biologically inspired or spiking neural networks for reactive control [18, 19], as well as hybrid adaptive models based on extended Kalman filtering [20] or learned barrier functions for safe navigation [21].

Despite these advances, most current solutions rely on predefined reference signals, and relatively few studies tackle the dynamic pursuit of a moving virtual point without trajectory generation. The real-time steering of a DDMR toward a continuously changing virtual target represents a distinct and underexplored control paradigm, one that prioritizes reactivity, adaptability, and computational efficiency over long-horizon planning. Only a handful of studies, such as those of Yıldırım and Savaş [10] and Vo et al. [22], approached neural control in reactive contexts, yet without focusing on continuously moving goal points or real-time generalization to unseen motion patterns.

Building upon these insights, the present study introduces a Guidance Neural Network (GNN) architecture designed to steer a differential-drive mobile robot directly toward a moving virtual target. The novelty of this work lies in:

1. reformulating the control problem as real-time guidance rather than path tracking,
2. employing artificial potential field–based data generation for neural training, enabling the controller to learn reactive steering behaviors,
3. demonstrating robust generalization to unseen target trajectories and velocities in simulation benchmarks.

The remainder of this paper is structured as follows. Section 2 discusses the theoretical background and control model. Section 3 presents the proposed GNN design and training methodology. Section 4 details the simulation setup, and section 5 concludes with insights into the system's performance, adaptability, and prospects for real-time implementation on embedded robotic platforms.

2. CONTROL PROBLEM AND TRAINING DATA ACQUISITION

The first step in designing the artificial neural network (GNN) is the preparation of a dataset that will be used to train the network. Therefore, a position-tracking reference algorithm must be implemented to collect signal values that will serve as the neural network's inputs and outputs, corresponding respectively to the outputs of the measurement system and the inputs to the control system. The reference position-tracking algorithm can be realized either on a physical robot or through simulation models. Since the training dataset should encompass as many scenarios as possible that may occur during an actual robot mission, a large number of trials must be performed. Consequently, acquiring the required data through simulation constitutes a more practical and efficient approach.

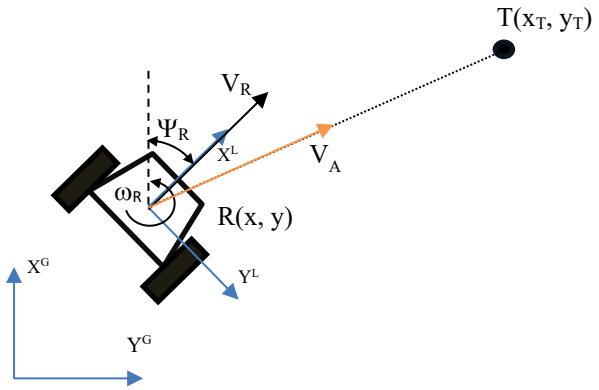

Fig. 1. Differential drive robot *R* attracted to the target *T* by applying velocity vector $\vec{V}_A$

To create a numerical model of position tracking control, the artificial potential field approach was applied, which applies only an attraction force. In future work, it will be extended with repulsion for obstacle avoidance by the robot. The principle of attraction is presented in Figure 1, where $\vec{V}_A$ is a velocity vector related with attraction to the target.

Velocity vector $\vec{V}_A$ in artificial potential field approach is just reverse gradient of potential field defined as follows:

$$U(x,y) = (x - x_T)^2 + (y - y_T)^2 \tag{1}$$

where: x, y – robot's coordinates in a global coordinate frame *G*, x_T, y_T – target's coordinates in the global coordinate frame *G*.

The point of minimum of the function $U(x,y)$ is located at the target's position, i.e., $T(x_T, y_T)$, and gradient of the potential function at robot's position is $R(x,y)$ in global reference frame *G*:

$$\nabla\vec{U}(x.y) = \begin{bmatrix} \frac{\delta U}{\delta x} \\ \frac{\delta U}{\delta y} \end{bmatrix} = \begin{bmatrix} 2\cdot(x - x_T) \\ 2\cdot(y - y_T) \end{bmatrix} \tag{2}$$

If the velocity vector is reverse to the gradient, it can be defined as follows:

$$\vec{V}_A = -\nabla\vec{U}(x.y) = \begin{bmatrix} -2\cdot(x - x_T) \\ -2\cdot(y - y_T) \end{bmatrix} \tag{3}$$

Having vector $\vec{V}_A$ allows us to determine control signals required to guide the robot towards the target, i.e. linear speed V_R and angular rate ω_R. To determine this, vector $\vec{V}_A$ must be rotated from the global coordinate frame *G* to robot's body frame *L* about Ψ_R. The rotation is defined as follows:

$$\vec{V}_A^L = \begin{bmatrix} cos(\Psi_R) & sin(\Psi_R) \\ -sin(\Psi_R) & cos(\Psi_R) \end{bmatrix} \cdot \vec{V}_A \tag{4}$$

where: Ψ_R – is the current heading angle of the robot, $\vec{V}_A^L$ – is the desired velocity vector in the frame *L*.

The desired velocity vector $\vec{V}_A^L$ given in local coordinate frame is used to determine control signals:

$$V_R = \begin{cases} k_V \cdot \left|\vec{V}_A^L\right|;\ k_V \cdot \left|\vec{V}_A^L\right| \le V_{max} \\ V_{max};\quad k_V \cdot \left|\vec{V}_A^L\right| > V_{max} \end{cases} \tag{5}$$

where: V_{max} – is the maximum admissible speed of the robot, k_V – proportional gain coefficient.

$$\omega_R = \frac{k_\omega}{\pi} \cdot \left(atan2\left(\frac{\vec{V}_A(y)}{\vec{V}_A(x)}\right) - \Psi_R\right) \tag{6}$$

where: k_ω – is a proportional gain coefficient.

The linear velocity $V_R$represents a scaled magnitude of the desired velocity vector $\vec{V}_A^L$, constrained by the robot's maximum admissible linear speed. Conversely, the angular velocity ω_R is generated by a proportional controller, in which the control error corresponds to the heading angle deviation – that is, the difference between the current and the desired heading angle of the robot. Finally, we can get robot's control vector, which will be produced by neural network:

$$\vec{u} = \begin{bmatrix} V_R \\ \omega_R \end{bmatrix} \tag{7}$$

where: V_R – linear speed input, ω_R – angular rate input.

Let's consider standard kinematic model for differential drive mobile robot in its body frame [23]:

DOI: 10.65731/ama/2026-0067

$$\begin{bmatrix} \dot{x}^L \\ \dot{y}^L \\ \dot{\Psi}_R \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \cdot \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \tag{8}$$

where: r – wheels radius, b – width of the robot, x^L, y^L – robot's coordinates in the body frame, Ψ_R – heading angle, ω_l, ω_r – angular rates respectively of left and right wheel.

In the global reference frame, the robot's kinematics can be expressed as follows:

$$\begin{bmatrix} \dot{x}^G \\ \dot{y}^G \\ \dot{\Psi}_R \end{bmatrix} = \begin{bmatrix} \cos(\Psi_R) & 0 \\ \sin(\Psi_R) & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_R \\ \omega_R \end{bmatrix} = \begin{bmatrix} \cos(\Psi_R) & \sin(\Psi_R) & 0 \\ -\sin(\Psi_R) & \cos(\Psi_R) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{x}^L \\ \dot{y}^L \\ \dot{\Psi} \end{bmatrix} \tag{9}$$

Substituting equation (8) into (9) yields:

$$\begin{bmatrix} \dot{x}^G \\ \dot{y}^G \\ \dot{\Psi}_R \end{bmatrix} = \begin{bmatrix} \cos(\Psi_R) & \sin(\Psi_R) & 0 \\ -\sin(\Psi_R) & \cos(\Psi_R) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \cdot \begin{bmatrix} \omega_l \\ \omega_r \end{bmatrix} \tag{10}$$

The angular velocities of the wheels can be obtained from the equations of the differential-drive controller [23] as follows:

$$\omega_l = \frac{V_R - \omega_R \cdot \frac{b}{2}}{r} \tag{11}$$

$$\omega_r = \frac{V_R + \omega_R \cdot \frac{b}{2}}{r} \tag{12}$$

The final kinematic model, which can be applied in numerical simulations to generate data for neural network training, is expressed as follows:

$$\begin{bmatrix} \dot{x}^G \\ \dot{y}^G \\ \dot{\Psi}_R \end{bmatrix} = \begin{bmatrix} \cos(\Psi_R) & \sin(\Psi_R) & 0 \\ -\sin(\Psi_R) & \cos(\Psi_R) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ 0 & 0 \\ -\frac{r}{b} & \frac{r}{b} \end{bmatrix} \cdot \begin{bmatrix} \frac{V_R - \omega_R \cdot \frac{b}{2}}{r} \\ \frac{V_R + \omega_R \cdot \frac{b}{2}}{r} \end{bmatrix} \tag{13}$$

The model described by equation (13) was used to simulate a series of robot motion trajectories, in which the virtual point moved along four different path shapes and at varying speeds. The geometrical parameters of the model, as well as the limits for linear velocity and angular rate, were set according to the technical specifications of the *TurtleBot3* platform as framework for further experimental verification (Table 1).

Tab. 1. *TurtleBot3* parameters

Parameter	Value
Wheel radius *r*	0.033 m
Vehicle width *b*	0.16 m
Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s

* https://emanual.robotis.com/docs/en/platform/turtlebot3/features/

As the generated trajectories serve as reference data for training the neural network, employing a kinematic rather than a dynamic model ensures reduced sensitivity of the network to modeling uncertainties. Four distinct trajectories of the virtual point were designed for the tracking task, namely rectangular, triangular, circular, and figure-eight paths. The virtual point was assigned the following velocity magnitudes: 0.05, 0.075, 0.10, 0.125, 0.15, and 0.175 m/s. Different starting point was selected for each trial.

Figures 2÷5 illustrate the trajectories of the moving virtual point together with the simulated paths followed by the robot. To provide more readability of figures, plots are limited to speeds: 0.05, 0.10 and 0.15 m/s.

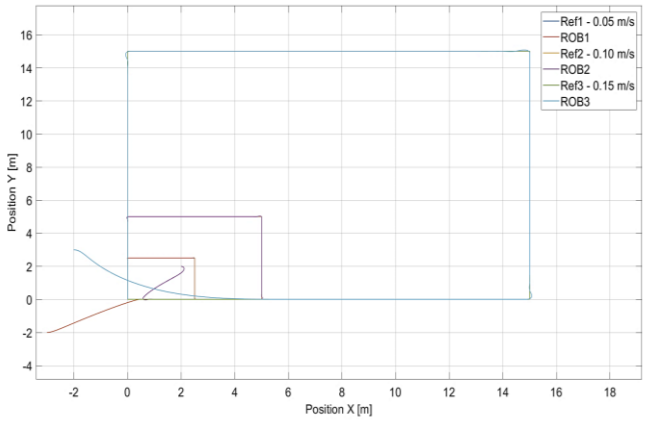

Fig. 2. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.05, 0.10, and 0.15 m/s – rectangular shape

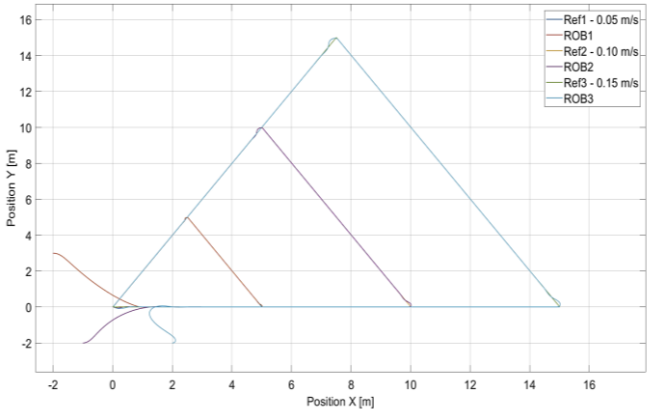

Fig. 3. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.05, 0.10, and 0.15 m/s – triangular shape

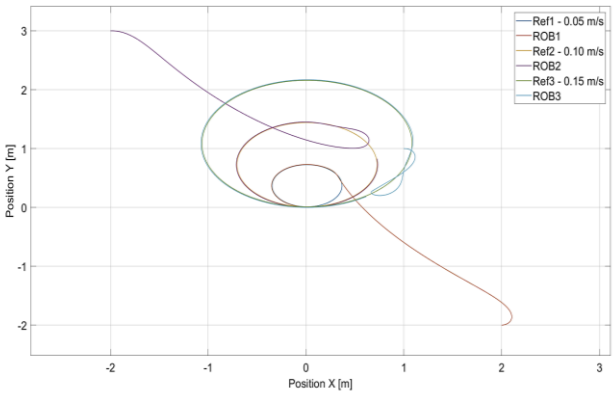

Fig. 4. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.05, 0.10, and 0.15 m/s – circular shape

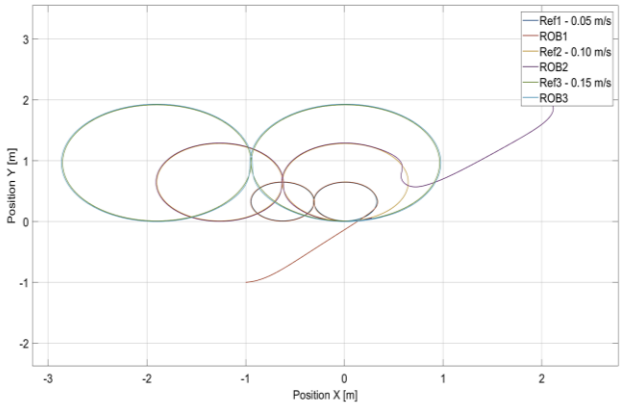

Fig. 5. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.05, 0.10, and 0.15 m/s – eight shape

In the conducted simulations, the gain coefficients k_ω and k_V were determined to be 3.2 and 0.5, respectively without their optimization. The trajectory plots presented in Figures 2÷5 demonstrate that the robot tracked the virtual reference point smoothly, exhibiting no significant overshoot during turning maneuvers.

For the purpose of generating training data from the simulations, the input and output vectors for the neural network were defined. The output vector of the neural network corresponds to the control vector $\vec{u}$ (Eq. 7). The range of values of $\vec{u}$ is narrower than the interval [0, 1], which represents the typical output range producible by the neural network. For the input vector, signals containing appreciable information about the control of trajectory tracing should be used and next scaled to the range [0, 1]. The following information was used to define the input for the neural network:

1. relative position of the robot with respect to the position of the virtual point,
2. actual heading angle of the robot,
3. tracking error defined as the distance between the robot and the virtual point,
4. rate of change of the robot's heading angle.

Thus, the input vector containing the listed information, scaled to the range [0, 1], can be defined as follows:

$$U_{NN} = \begin{bmatrix} \frac{x-x_T}{\sqrt{(x-x_T)^2+(y-y_T)^2}} \\ \frac{y-y_T}{\sqrt{(x-x_T)^2+(y-y_T)^2}} \\ \frac{\Psi_R}{\pi} \\ \frac{\sqrt{(x-x_T)^2+(y-y_T)^2}}{D_{max}} \\ \frac{d\Psi_R}{dt} \end{bmatrix} = \begin{bmatrix} \frac{x-x_T}{D} \\ \frac{y-y_T}{D} \\ \frac{\Psi_R}{\pi} \\ \frac{D}{D_{max}} \\ \frac{d\Psi_R}{dt} \end{bmatrix} \tag{14}$$

where: U_{NN} – the neural network input vector, x, y – coordinates of the robot's position, x_T, y_T – coordinates of the virtual point, Ψ_R – the actual heading angle of the robot, D – the tracking error, D_{max} – the maximum value of the tracking error.

In turn, the neural network output vector can be expressed as:

$$Y_{NN} = \begin{bmatrix} V_{NN} \\ 2.84 \cdot \omega_{NN} \end{bmatrix} \tag{15}$$

where: V_{NN} – denotes the linear velocity obtained from the neural network output, ω_{NN} – represents the angular velocity derived from the neural network output.

The first two elements of U_{NN} correspond to the components of a vector defined by the positions of the robot and the virtual point, normalized to a unit length. The next input represents the robot's heading angle, expressed in the half-compass system and scaled to the range [0, 1] in radians. The fourth input is the tracking error, defined as the distance between the robot and the virtual point, divided by its maximum value to ensure normalization to the range [0, 1]. The last input is the rate of change of the heading angle, which due to the capabilities of the *TurtleBot3* robot is inherently limited to the range [0, 1] rad/s.

In the case of the output vector (Eq. 15), it should be recalled that the neural network outputs are constrained to the range [0, 1.0] for linear speed and [-1.0,1.0] for angular speed. Therefore, to ensure compatibility between the neural network's output vector and the robot's input vector, the values must be scaled according to the maximum limits of the robot's inputs. Since the robot's linear velocity already lies within the range [0, 1.0], only the angular velocity needs to be scaled by a factor of 2.84 rad/s to achieve range [-2.84, 2.84].

3. GUIDANCE NEURAL NETWORK DESIGN

The designed guidance neural network must balance two inherently conflicting objectives: achieving precise virtual point tracking while remaining simple enough for implementation on real differential-drive robots. The first objective implies that the tracking error should remain minimal at every point along the robot's trajectory and should be, at least, comparable to that observed in the training dataset. The magnitude of tracking errors in the guidance neural network depends exclusively on the quality of the training dataset rather than on the network architecture itself. Therefore, this aspect is not analyzed further, as the gain coefficients of the reference control method used to generate the training data were not optimized. Instead, the teaching and testing errors are discussed as quality indicators for comparing different network configurations.

The second objective concerns the computational efficiency necessary for real-time execution of the neural network on the robot's onboard controller. In previous research addressing a similar problem, a comparison of computational performance across commonly used neural network architectures was conducted namely, a multilayer perceptron (MLP) with a single hidden layer, an MLP with two hidden layers, a residual neural network, and deep neural networks [24]. The reported results indicated that deep neural networks achieved the lowest teaching and testing errors, whereas the single-hidden-layer MLP exhibited the highest computational efficiency. Notably, the error levels of multilayer perceptron networks were only marginally higher than those of deep neural networks and verification them through simulation was successful. Therefore, it can be anticipated that, in the case of the virtual point tracking problem applied to a differential-drive robot, a MLP with a single hidden layer will also be suitable for this task. The hypothetical structure of this network is shown in Fig. 6.

Designing a guidance neural network (GNN) based on a single-layer multilayer perceptron (MLP) involves determining a network architecture that minimizes teaching and testing errors while maximizing the correlation between the network outputs and the reference outputs from the training dataset. The optimal number of neurons in the hidden layer and the specific forms of the activation functions are generally unknown *a priori* and must be identified during the training phase as part of defining the most suitable neural function structure. The training of the single-layer MLP was conducted

DOI: 10.65731/ama/2026-0067

using *Statistica* software where the BFGS method (Broyden-Fletcher-Goldfarb-Shanno algorithm) is applied, and the corresponding results are summarized in Tab. 2. The acronym used to denote specific network configurations follows the convention MLP I_H_O, where *I* = *5* represents the number of inputs (as defined in Eq. 14), *O* = *2* denotes the number of outputs (as defined in Eq. 7), and *H* indicates the number of neurons in the hidden layer, which varies among the networks examined during the training process. The hidden layer size was automatically determined by *Statistica*, which randomly selected the number of neurons within the range of 100 to 400. Likewise, the activation functions for the hidden and output layers were randomly chosen from the following set: *sinus*, *tanh*, *exponential*, *identity*, and *logistic*. Altogether, this gives 7500 combinations, and Tab. 2 presents the results, limited to 20 neural networks. Investigating all 7500 combinations will take an extremely long time. With green color, the best neural network was indicated and with red – the worse one.

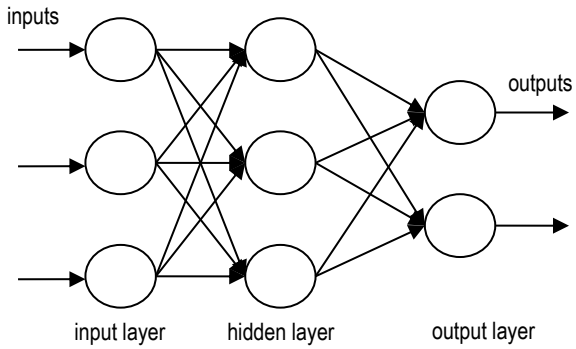

Fig. 6. The hypothetical structure of this network

Tab. 2. Comparison of training and testing results for different structures of MLP neural networks

MLP network	Teaching corr.	Testing corr.	Teaching error	Testing error	Activation function hidden	Activation function output
MLP 5_370_2	0.926454	0.928143	0.005497	0.005423	Sinus	Logistic
MLP 5_279_2	0.999768	0.999573	0.000010	0.000018	Tanh	Tanh
MLP 5_318_2	0.999420	0.999255	0.000019	0.000025	Logistic	Exponential
MLP 5_226_2	0.999778	0.999764	0.000007	0.000007	Exponential	Logistic
MLP 5_186_2	0.724654	0.724433	0.007103	0.007015	Sinus	Sinus
MLP 5_108_2	0.752696	0.751707	0.004274	0.004353	Sinus	Identity
MLP 5_398_2	0.999212	0.999127	0.000020	0.000023	Tanh	Tanh
MLP 5_341_2	0.999579	0.998193	0.000014	0.000075	Logistic	Logistic
MLP 5_356_2	0.682353	0.688430	0.002535	0.002477	Sinus	Exponential
MLP 5_345_2	0.998624	0.998173	0.000036	0.000055	Tanh	Identity
MLP 5_160_2	0.769018	0.768935	0.001742	0.001765	Sinus	Identity
MLP 5_364_2	0.999502	0.999410	0.000016	0.000019	Exponential	Logistic
MLP 5_297_2	0.999254	0.999034	0.000018	0.000027	Exponential	Tanh
MLP 5_160_2	0.974112	0.974792	0.001070	0.001028	Sinus	Tanh
MLP 5_341_2	0.905324	0.901360	0.003785	0.003850	Exponential	Exponential
MLP 5_226_2	0.926454	0.928143	0.005497	0.005423	Sinus	Logistic
MLP 5_314_2	0.999886	0.999845	0.000003	0.000004	Logistic	Logistic
MLP 5_221_2	0.999816	0.999766	0.000006	0.000008	Logistic	Logistic
MLP 5_236_2	0.999384	0.999171	0.000019	0.000023	Logistic	Logistic
MLP 5_177_2	0.999362	0.999053	0.000022	0.000029	Logistic	Logistic
MLP 5_289_2	0.998868	0.998762	0.000029	0.000031	Logistic	Logistic

Analysis of the tabulated results indicates that the single-layer MLP architecture employing logistic activation functions in both the hidden and output layers consistently outperforms networks using alternative activation schemes, with the tanh function yielding only marginally inferior performance. Furthermore, no consistent pattern or theoretical rationale suggests that increasing the number of neurons invariably leads to improved performance, reduced error metrics, or higher correlation coefficients. According to Tab. 2, the best-performing neural network is the MLP 5_314_2, which is noticeably superior to the MLP 5_341_2 model despite sharing the same activation function and having higher number of neurons. It is important to note that the training process is inherently an optimization problem, in which finding a global minimum is challenging due to the presence of multiple local minima.

The computational cost of the proposed controller is determined by the forward-pass evaluation of the fully connected MLP 5_314_2 network. For a multilayer perceptron containing n_{in} inputs, n_{h} neurons in the hidden layer, and n_{out} outputs, the time complexity of a single inference step is:

$$\mathcal{O}(n_{\mathrm{in}} n_{\mathrm{h}} + n_{\mathrm{h}} n_{\mathrm{out}}) \tag{16}$$

For the selected network, $n_{\mathrm{in}} = 5$, $n_{\mathrm{h}} = 314$, and $n_{\mathrm{out}} = 2$. Consequently, one forward pass involves:

$$5 \cdot 314 + 314 \cdot 2 = 2198 \tag{17}$$

weight multiplications, approximately 2198 additions including the bias terms, and 316 activation-function evaluations. The total number of trainable parameters, including weights and biases, is:

$$P = (5 + 1) \cdot 314 + (314 + 1) \cdot 2 = 2514 \quad (18)$$

When represented using 32-bit floating-point numbers, the network parameters require approximately 9.82 KiB of memory. The BFGS optimization algorithm is applied exclusively during the offline training stage and therefore does not contribute to the computational cost of controller inference. During the simulation, only the forward-pass calculation is performed. For the fixed MLP 5_314_2 architecture, the computational cost of each control step is constant and independent of the duration of the simulated robot mission.

The relatively limited number of arithmetic operations indicates that the selected architecture may be suitable for future implementation on embedded robotic platforms. However, platform-dependent execution-time measurements and real-time experimental validation remain outside the scope of the present simulation-based study.

As indicated in previous work [24], even excellent results achieved by the neural network do not guarantee that it will be capable of controlling the robot's position in the expected manner. Therefore, it is essential to validate the network experimentally, at minimum through numerical simulation. The next section presents the simulation results, in which the neural network is employed as the controller governing the robot's motion. Those results will be compared with the reference method used to generate the training dataset.

4. RESULTS OF NUMERICAL SIMULATIONS

To explore the capabilities of the designed GNN and to validate the outcomes obtained during training trials, a series of numerical simulations were performed. In these simulations, the same path geometries were employed; however, the velocities of the virtual point differed from those used in the training dataset. The robot's initial position was randomized, while the initial heading angle was fixed at 0^0. The starting position of the virtual point was set to [0, 0]. Figures 7÷10 present simulation result, i.e., achieved path of the robot, which tracks the virtual point with the use of MLP 5_314_2 network as GNN controller.

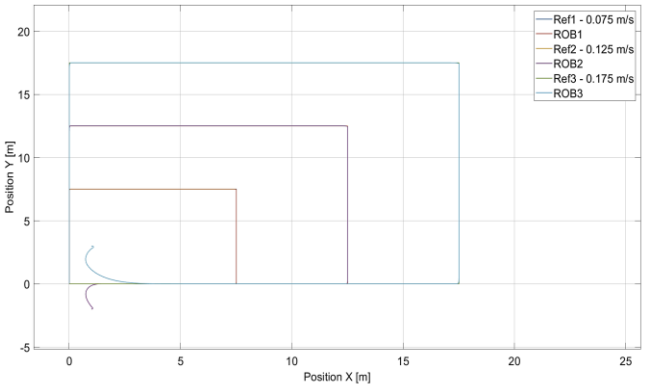

Fig. 7. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.075, 0.125, and 0.175 m/s – rectangle shape, with the use of MLP 5_314_2 neural network as GNN controller

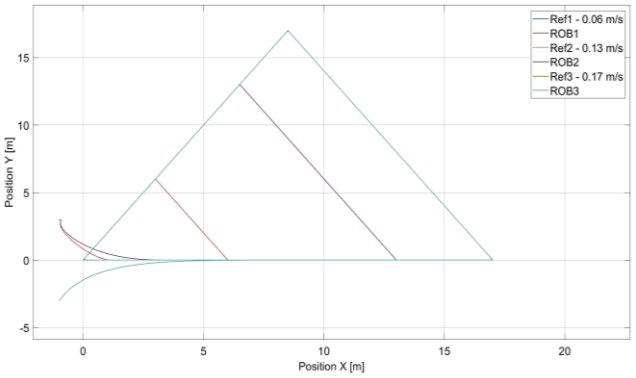

Fig. 8. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.06, 0.13, and 0.17 m/s – triangle shape, with the use of MLP 5_314_2 neural network as GNN controller

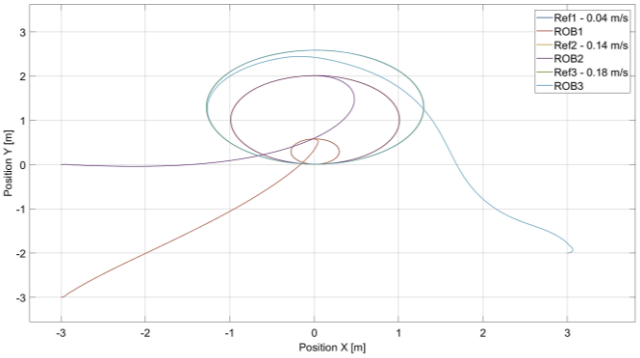

Fig. 9. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.04, 0.14, and 0.18 m/s – circular shape, with the use of MLP 5_314_2 neural network as GNN controller

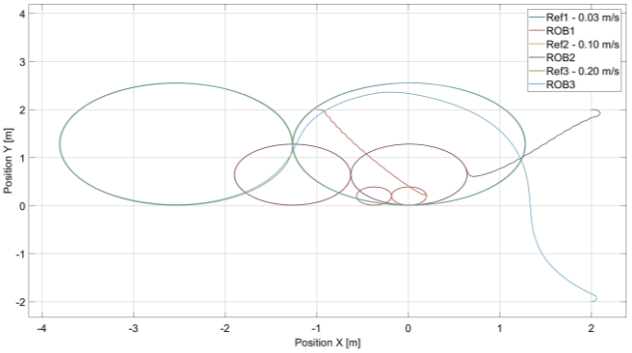

Fig. 10. Virtual point (Ref) and robot (ROB) trajectories for three different speeds of virtual point: 0.03, 0.10, and 0.20 m/s – eight shape, with the use of MLP 5_314_2 neural network as GNN controller

Based on the figures illustrating the robot's trajectories, it can be observed that the GNN controller successfully managed the tracking process smoothly, without significant errors or overshoots. The shapes of the robot's trajectories are not a decisive factor in determining the effectiveness of the GNN controller. Therefore, a quantitative assessment was performed by defining a parameter represented by the root mean square error ($RMSE_{TE}$) of the tracking error, which is a distance between the robot and the virtual point (equation 19).

$$RMSE_{TE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left(\sqrt{(x_{Ri} - x_{Ti})^2 + (y_{Ri} - y_{Ti})^2}\right)^2} \quad (19)$$

DOI: 10.65731/ama/2026-0067

where: x_{Ri}, y_{Ri} – robot's position in the i^{th} moment, x_{Ti}, y_{Ti} – virtual point's position in the i^{th} moment, N – number of samples in simulations.

Table 3 summarizes the RMSETE values for both the GNN controller and the Artificial Potential Field (APF) controller as reference method across all shape cases, considering different virtual point speeds and initial robot positions.

Tab. 3. Comparison of tracking error RMSE value between APF and GNN methods for different shapes and speeds

Shape	Speed [m/s]	Robot's initial position		Tracking error $RMSE_{TE}$ - APF	Tracking error $RMSE_{TE}$ - GNN
		X	Y		
Rectangular	0.075	0	0	0.02125	0.01697
	0.125	1	-2	0.06869	0.06353
	0.175	1	3	0.1092	0.1097
Triangular	0.06	-1	3	0.1094	0.1112
	0.13	-1	3	0.1285	0.1317
	0.17	-1	-3	0.1539	0.1494
Circular	0.04	-3	-3	0.1699	0.1688
	0.14	-3	0	0.1356	0.1337
	0.18	3	-2	0.143	0.136
Figure-eight	0.03	-1	2	0.06245	0.06183
	0.10	2	2	0.08382	0.08045
	0.20	2	-2	0.1251	0.1171

The $RMSE_{TE}$ values for the APF method are slightly lower only in three cases, indicating superior tracking efficiency compared to the GNN controllers in those instances. However, in most of the cases presented in Tab. 3, the GNN outperforms the APF approach. Considering that the APF method was used to generate the training dataset for the GNN, these results are somewhat unexpected. Moreover, the coefficients of the APF controller were not optimized to maximize its effectiveness; therefore, the observed performance improvement appears to arise from the GNN controller itself, even though neural network capabilities are inherently constrained by the quality of the training data. Expanding the training dataset—both in size and diversity—could potentially further enhance the GNN's performance.

The results presented in Tab. 3 are consistent with the trends shown in figures 11÷14, where the GNN achieves lower steady-state tracking errors. During the first 25 seconds of simulation, when the initial distance between the robot and the virtual target point is relatively large the APF-based controller is able to reduce the tracking error more rapidly than the GNN, as observed in Fig. 11. This behavior influences the $RMSE_{TE}$ values reported in Tab. 3, but not for all cases. The faster initial response of the APF may stem from the lower desired linear velocities generated by the GNN.

However, the principal advantage of employing a GNN controller lies in its ability to be trained beyond the specific cases and environmental conditions considered in analytical methods. This capability becomes particularly important when the system is exposed to disturbances or other stochastic and unpredictable factors, where the extrapolation potential of artificial neural networks can provide improved robustness and improvement of efficiency. From this perspective, the obtained results are highly promising and support the suitability of neural networks for trajectory-tracking tasks in robotic systems.

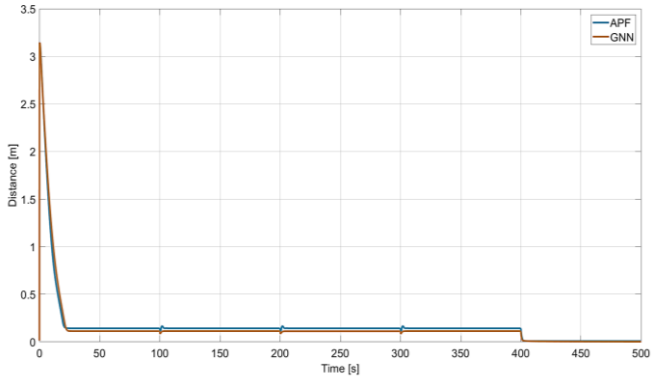

Fig. 11. Time plots of tracking error for GNN and APF – rectangular shape and 0.15 m/s speed

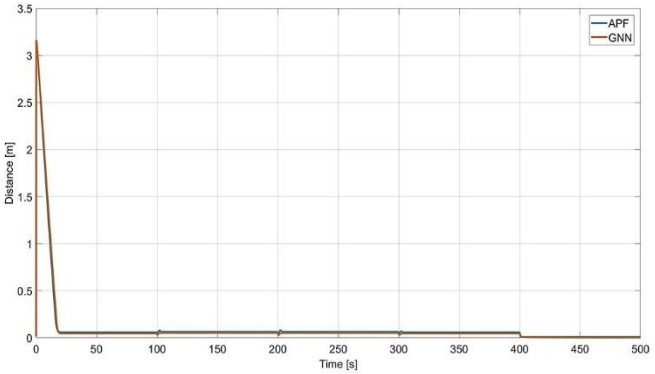

Fig. 12. Time plots of tracking error for GNN and APF – triangular shape and 0.06 m/s speed

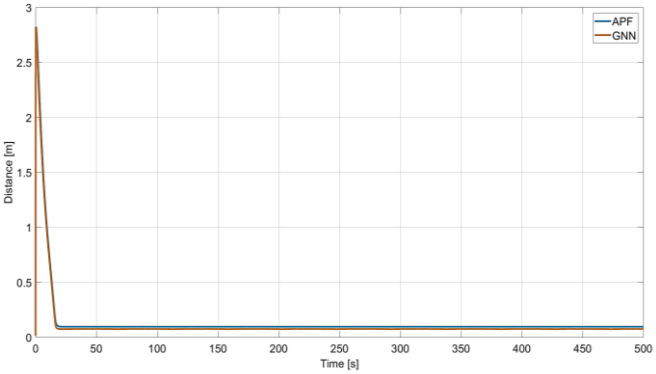

Fig. 13. Time plots of tracking error for GNN and APF – circular shape and 0.10 m/s speed

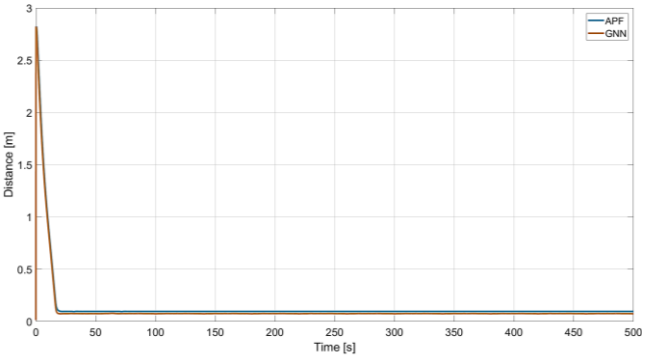

Fig. 14. Time plots of tracking error for GNN and APF – figure-eight shape and 0.10 m/s speed

Thus, since the GNN is trained on a dataset that excludes measurement noise from the robot's onboard sensors and positioning system, it is necessary to examine how it performs under noisy conditions. Measurement noise was simulated as possible realistic measurement interference from sensors. White noise with flat power spectral density (PSD) was added, resulting in maximum variations magnitude of ±0.02 m in position and ±0.01 rad in heading. Figure 15 presents simulation results in which the robot tracks a virtual point along a triangular reference trajectory at a speed of 0.04 m/s with noises. It can be observed that, locally, the robot is unable to move precisely along straight segments; however, globally, it still manages to follow the reference trajectory. Issues related to noise, positioning errors, and phenomena such as wheel slip affect any guidance algorithm that does not employ advanced SLAM techniques.

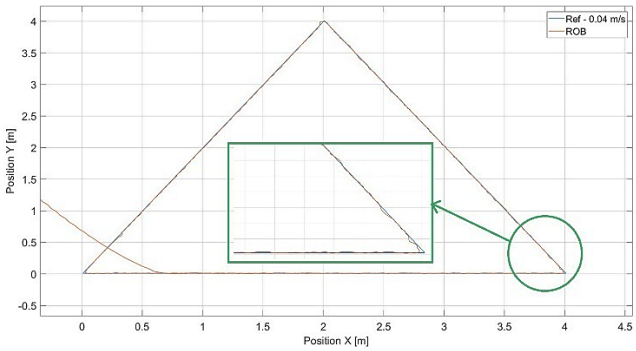

Fig. 15. Virtual point (Ref) and robot (ROB) trajectories at the speed of virtual point 0.04 m/s – triangle shape, with the use of MLP 5_314_2 neural network as GNN controller and including measurement noise of robot's position and heading angle

A training dataset can be constructed by combining measurement data collected under noisy conditions with control signals corresponding to noise-free cases for identical reference trajectories and speeds. If a neural network capable of learning from such data can be developed, the GNN may inherently gain filtering capabilities. This will be investigated in future research.

5. CONCLUSIONS

This study presented the design, training, and evaluation of a Guidance Neural Network (GNN) intended to directly control a differential-drive mobile robot and guide it toward a moving virtual target without relying on traditional control loops or predefined trajectories. The artificial potential field (APF) method was employed solely as a reference mechanism to generate a diverse training dataset comprising multiple trajectory geometries and virtual-target velocities. Based on the training outcomes, a single-hidden-layer multilayer perceptron (MLP) structure was identified as the most efficient architecture, providing an advantageous balance between computational simplicity and prediction accuracy.

Simulation results demonstrated that the GNN controller is capable of achieving smooth and stable tracking performance for all test scenarios, including path shapes and speeds not present in the training dataset. Quantitative evaluation confirmed that, in most cases, the GNN achieved lower or comparable RMSE tracking errors relative to the APF controller used to generate the training set. This is particularly notable given that the APF controller coefficients were not optimized. Although the GNN was trained using control signals generated by the APF method, its closed-loop behavior does not have to reproduce the APF response exactly. Approximation errors, nonlinear activation functions, and output scaling may modify the generated control signals and consequently lead to different tracking-error characteristics. Nevertheless, the GNN exhibited an inherent ability to generalize and maintain robust behavior under varying initial conditions, demonstrating its capacity to reproduce reactive steering behavior learned from the APF method.

Analysis of time-domain error plots further revealed that while the APF controller may reduce large initial tracking errors more rapidly in some cases, the GNN provided superior steady-state performance. This suggests that neural controllers, despite their limited interpretability, can internalize advantageous control behaviors while smoothing out abrupt responses typical of purely analytical approaches.

Overall, the simulation results demonstrate the potential of a properly trained GNN as a direct controller for virtual-target pursuit in differential-drive robots. Experimental real-time validation is required before definitive conclusions regarding its performance on a physical robotic platform can be formulated. Crucially, its generalization capability beyond the training distribution underscores the potential of neural networks for navigation subject to stochastic disturbances, nonlinear dynamics, and unmodeled interactions—precisely the conditions that challenge classical methods. Future research will, therefore, focus on integrating the proposed GNN into a hierarchical control structure, where a higher-level network or perception module dynamically selects target coordinates, with the ultimate goal of enabling fully autonomous, bio-inspired robot guidance.

REFERENCES

1. Fierro R, Lewis FL. Control of a nonholonomic mobile robot: back-stepping kinematics into dynamics. In: Proceedings of the 1995 34th IEEE Conference on Decision and Control. New Orleans, LA, USA. 1995; 4: 3805-3810. https://doi.org/10.1109/CDC.1995.479190
2. Nguyen T, Le L. Neural network-based adaptive tracking control for a nonholonomic wheeled mobile robot with unknown wheel slips, model uncertainties, and unknown bounded disturbances. Turk J Elec Eng Comput Sci. 2018;26(1):378-392. https://doi.org/10.3906/elk-1705-167
3. Cardona M, Serrano FE. Dynamic output feedback and neural network control of a non-holonomic mobile robot. Sensors (Basel). 2023;23(15):6875. https://doi.org/10.3390/s23156875
4. Boukens M, Boukabou A, Chadli M. Robust adaptive neural network-based trajectory tracking control approach for nonholonomic electrically driven mobile robots. Robot Auton Syst. 2017;92:30-40. https://doi.org/10.1016/j.robot.2017.03.001
5. Szuster M, Szeremeta M. Neural tracking control of a four-wheeled mobile robot with Mecanum wheels. Appl Sci. 2022;12(11):5322. https://doi.org/10.3390/app12115322
6. Trinh TK Ly, Nguyen HT, Luu TP. Design of neural network-PID controller for trajectory tracking of differential drive mobile robot. Vietnam J Sci Technol. 2024;62(2):374-386. https://doi.org/10.15625/2525-2518/18066
7. Hu T, Yan SX, Wang F, Mittal GS. A neural network controller for a nonholonomic mobile robot with unknown robot parameters. In: Proceedings of the 2002 IEEE International Conference on Robotics and Automation. 2002; 4: 3540-3545. https://doi.org/10.1109/ROBOT.2002.1014258
8. Liu H, Li Z, Wang H, Zhang J. Neural network-based adaptive motion control for a mobile robot with unknown longitudinal slipping. Chin J Mech Eng. 2019;32(1):1-9. https://doi.org/10.1186/s10033-019-0373-3
9. Lopez-Franco M, Sánchez EN, Alanis AY, López-Franco C. Neural

DOI: 10.65731/ama/2026-0067

control for a differential drive wheeled mobile robot integrating stereo vision feedback. Comput Sist. 2015;19(3):429-443. https://doi.org/10.13053/CyS-19-3-2016
10. Yıldırım Ş, Savaş S. Tracking control of a nonholonomic mobile robot using neural network. In: Maki K, editor. Handbook of research on advancements in robotics and mechatronics. Hershey (PA): IGI Global. 2015; 631-661. https://doi.org/10.4018/978-1-4666-7387-8.ch028
11. Gulo SC, Tamba T. Design of trajectory tracking control for a differential drive wheeled mobile robot. J Nat Tek Elektro Tek Inform. 2021;10(3):272-281. https://doi.org/10.22146/jnteti.v10i3.1454
12. Juhi HH, Ameen NM, Mahmood SA, Mohammed YA, Yahya AA. Robust neural network-based trajectory tracking control for mobile vehicles. J Intell Syst Control. 2024;3(4):251-259. https://doi.org/10.56578/jisc030405
13. Sutton RS, Barto AG. Reinforcement learning: an introduction. 2nd ed. Cambridge (MA): MIT Press; 2018.
14. Hongyi L. Mobile robot navigation based on deep reinforcement learning: a brief review. J Phys Conf Ser. 2023;2649(1):012027. https://doi.org/10.1088/1742-6596/2649/1/012027
15. Yang G, Guo Y. Deep reinforcement learning based mobile robot navigation in crowd environments. In: Proceedings of the 2024 21st International Conference on Ubiquitous Robots (UR). 2024; 513-519. https://doi.org/10.1109/UR61395.2024.10597481
16. Zhu Y, Wan Hasan WZ, Harun Ramli HR, Norsahperi NMH, Mohd Kassim MS, Yao Y. Deep reinforcement learning of mobile robot navigation in dynamic environment: a review. Sensors (Basel). 2025;25(11):3394. https://doi.org/10.3390/s25113394
17. Altawaitan A, Stanley J, Ghosal S, Duong T, Atanasov N. Hamiltonian dynamics learning from point cloud observations for nonholonomic mobile robot control. In: Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA). Yokohama, Japan. 2024;16937-16944. https://doi.org/10.1109/ICRA57147.2024.10610395
18. Zahra O, Tolu S, Navarro-Alarcon D. Differential mapping spiking neural network for sensor-based robot control. Bioinspir Biomim. 2021;16(3):036008. https://doi.org/10.1088/1748-3190/abedce
19. Moeys D, Corradi F, Kerr E, Vance P, Das G, Neil D, et al. Steering a predator robot using a mixed frame/event-driven convolutional neural network. In: Proceedings of the Second International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP). Krakow, Poland. 2016;1-8. https://doi.org/10.1109/EBCCSP.2016.7605233
20. Silaa MY, Bencherif A, Barambones O. Indirect adaptive control using neural network and discrete extended Kalman filter for wheeled mobile robot. Actuators. 2024;13(2):51. https://doi.org/10.3390/act13020051
21. Lafmejani AS, Berman S. NMPC-LBF: nonlinear MPC with learned barrier function for decentralized safe navigation of multiple robots. In: Proceedings of the 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Kyoto, Japan. 2022;10297-10303. https://doi.org/10.1109/IROS47612.2022.9981177
22. Vo TH, Nguyen TT, Than TT, Bui HY. Trajectory tracking intelligent controller for differential wheel mobile robot. J Mil Sci Technol. 2023;90(90):11-21. https://doi.org/10.54939/1859-1043.j.mst.90.2023.11-21
23. Klančar G, Zdešar A, Blažič S, Škrjanc I, editors. Wheeled mobile robotics: from fundamentals towards autonomous systems. Oxford: Butterworth-Heinemann; 2017. ISBN: 978-0-12-804204-5.
24. Kownacki C, Romaniuk S, Derlatka M. Applying neural networks as direct controllers in position and trajectory tracking algorithms for holonomic UAVs. Sci Rep. 2025;15:12605. https://doi.org/10.1038/s41598-025-97215-9

The work has been accomplished under the research project No. WZ/WM-IIM/4/2023 financed by Bialystok University of Technology.

Cezary Kownacki: https://orcid.org/0000-0003-0968-8211

Mariusz Bogdan: https://orcid.org/0000-0002-9960-7210