Dynamic Adaptive Dynamic Window Approach

Matej Dobrevski and Danijel Skočaj, *Member, IEEE*

***Abstract*—Robust local navigation is a critical capability for any mobile robot operating in a real-world unstructured environment, especially when there are humans or other moving obstacles in the workspace. One of the most commonly used methods for local navigation is the dynamic window approach (DWA), which does not address the problem of dynamic obstacles and depends heavily on the settings of the parameters in its cost function. Thus, it is a static approach that does not adapt to the characteristics of the environment, which can change significantly. On the other hand, data-driven deep learning approaches attempt to adapt to the characteristics of the environment by predicting the appropriate robot motion based on the current observation. However, they cannot guarantee collision-free trajectories for unseen inputs. In this work, we combine the best of both worlds. We propose a neural network to predict the weights of the DWA, which is then used for safe local navigation. To address the problem of dynamic obstacles, the proposed method considers a short sequence of observations to allow the network to model the motion of the obstacles and adjust the DWA weights accordingly. The network is trained using proximal policy optimization in a reinforcement learning setting in a simulated dynamic environment. We perform a comprehensive evaluation of the proposed approach in realistic scenarios using range scans of real 3-D spaces and show that it outperforms both DWA and purely deep learning approaches.**

***Index Terms*—Deep learning, mobile robotics, navigation, obstacle avoidance, reinforcement learning (RL).**

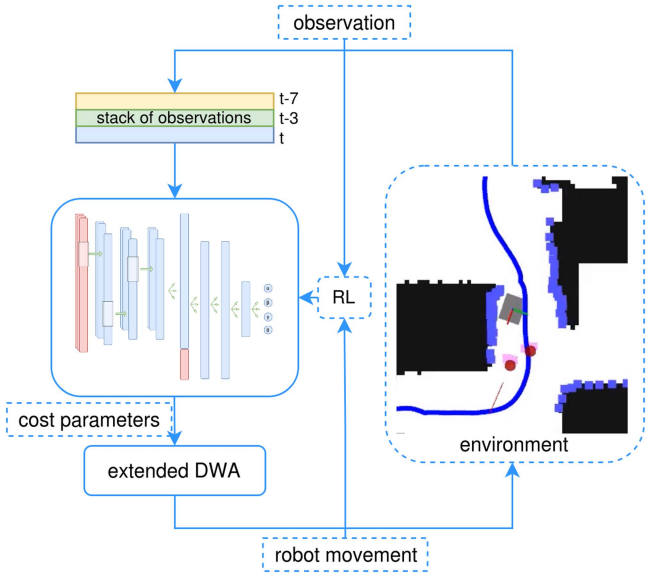

Fig. 1. Method for robot navigation in working areas populated with humans. Our method adapts the weights of an extended DWA algorithm by a neural network trained in an RL setting. The network adapts the weights on the basis of the viewpoint-aligned recent observations.

I. Introduction

THE ability to navigate unstructured environments without collisions is a crucial capability of mobile robots operating in the real world. Robust navigation is even more important, and at the same time more difficult to achieve, in dynamic environments with robots operating in the same working area as humans or other moving obstacles. This capability is typically implemented by utilizing a local planner, which generates movement commands for the robot while trying to simultaneously follow a global path and avoid humans and other obstacles.

One of the most commonly used local navigation methods is the dynamic window approach (DWA) [1]. It is well engineered, and considering the acceleration profile of the robot and the given distribution of obstacles around the robot, it produces movement commands that are guaranteed not to lead to collisions of the robot with the (static) environment. Due to its efficiency and good properties, it has been regularly used in many mobile robot navigation systems, including as the default local navigation method [2] in the navigation stack in the robot operating system (ROS) [3]. The DWA generates motor control commands in terms of translational and rotational velocities to achieve a goal position by using the dynamic model of the robot to optimize the distance to obstacles, its velocity, and the heading toward the goal. The reasons for longstanding and continuous usage of DWA are numerous: 1) it generates smooth trajectories; 2) the tradeoff between precision and computational power can be controlled via the sampling configuration; 3) it works in velocity space so there is no need for an additional controller; 4) and it guarantees collision-free trajectories in static environments. As such, it has been extended to numerous applications such as global navigation [4] and navigation in the presence of known moving obstacles [5]. Unfortunately, the DWA only takes into account static obstacles without considering the possible future positions of obstacles. Furthermore, the optimal settings of the weights in the cost function may vary; experimental results show that different weights are appropriate for different situations. Finally, the DWA needs a global path—and with that also a map of the environment, making it unsuitable for

Manuscript received 10 October 2023; revised 14 March 2024; accepted 5 April 2024. Date of publication 14 May 2024; date of current version 6 June 2024. This paper was recommended for publication by Associate Editor D. Panagou and Editor S. Behnke upon evaluation of the reviewers' comments. This work was supported in part by the Slovenian Research Agency through Research Project MV4.0 under Grant L2-3169 and through Research Program "Computer Vision" under Grant P2-0214. *(Corresponding author: Matej Dobrevski.)*

The authors are with the Faculty of Computer and Information Science, University of Ljubljana, SI-1001 Ljubljana, Slovenia (e-mail: matej.dobrevski@fri.uni-lj.si).

This article has supplementary downloadable material available at https://doi.org/10.1109/TRO.2024.3400932, provided by the authors.

Digital Object Identifier 10.1109/TRO.2024.3400932

usage in mapless scenarios. Therefore, while DWA has proven to be a very efficient engineering solution, it fails to adapt to the specifics of the individual environments and to the changes in the environment due to moving persons and other nonstationary obstacles.

In recent years, numerous approaches have been developed that try to tackle this problem in a data-driven way, by training a neural network to generate movement commands for the robot, either for static [6], [7], [8], [9], [10] or dynamic environments [11], [12]. However, a crucial issue in the use of neural networks for generating robot movement commands is the absence of any guarantees that the network will not generate a command that will cause a collision if, for example, an unforeseen input is fed to the network. In practice, the use of such a navigation system should be assisted with an additional system for verifying collisions, and if we want the robot to operate efficiently in dynamic environments, such a system would also need to take into consideration the dynamic parameters of the robot.

In this work, we address the issues of both rule-based and learning-based approaches and present a method that represents the best of both worlds. We design a neural network, which given the low-level perception of the distribution of obstacles around the robot produces optimal values for the weights in the DWA cost function. Our method dynamically adapts in a data-driven way, while utilizing the well-engineered DWA to produce safe and smooth commands. Furthermore, our method has no coupling with a global path, making it possible to perform in *mapless* scenarios. Our approach is illustrated in Fig 1.

We build on our previous adaptive DWA (ADWA) [13] method, by proposing an extension to the DWA cost function, considering a history of observations, and modifying the training settings for the neural network. The neural network is trained in a reinforcement learning (RL) manner, in simulated dynamic environments, using the proximal policy optimization (PPO) algorithm [14]. Experimentally, we show that this network does adapt the parameters of DWA according to the movement of obstacles and outperforms both the standard DWA implementation—our previous work, ADWA [13]—and recent related methods.

The contribution of this article is, therefore, a new *data-driven learning-based method for local robot navigation in dynamic environments* populated with moving pedestrians. As we will show in the following sections, the contribution is achieved by: 1) augmenting the traditional DWA cost function with an additional term that improves the performance of DWA given the recent history of observations and 2) the proposed deep-RL-based training approach. Since the proposed extension of DWA dynamically adapts to the changes in the environment, we termed it dynamic adaptive dynamic window approach (DADWA).

II. Related Work

Local navigation, or navigation based on the robot sensor readings, has been studied since the development of the first mobile robots. From these various approaches, the most used ones are methods operating in velocity space, as they consider the kinematic and dynamic constraints of the robot and, in contrast to geometrical approaches, can be easily translated to motor commands. Velocity space methods include several curvature-based methods [1], [15], [16], [17], from which the most popular is the DWA and is included as the default local navigation method in the ROS.

The DWA addresses the problem of navigation by searching, in the collision-free part of the reachable velocity space, for speeds that maximize a weighted cost function in order to maximize the clearance from obstacles, heading toward the goal, and movement speed of the robot. Brock and Khatib [4] expanded the algorithm so that it is applicable to global navigation scenarios. Their approach requires a costly calculation of a global navigation function and is not suitable for scenarios where the environment is expected to change. Ogren and Leonard [18] inspired by the work done by Rimon and Koditschek [19] develop a version of DWA that is provably convergent by viewing the method as model-predictive control.

Recently, Missura and Bannewitz [5] have presented a variant of DWA that also addresses navigation scenarios with moving obstacles, which are modeled as moving polygons, but it requires the knowledge of the movement of the polygons. Kobayashi and Motoi [20] expand the DWA with virtual manipulators to generate feasible trajectories among moving obstacles but require the knowledge of the kinematic parameters of the obstacles. Patel et al. [21] change the calculation of the DWA factors and train a neural network to select the best command instead of using the cost function. Pan et al. [22] propose a simple method for laser scan processing for segmenting obstacles and modeling their movement. This information was then used to generate trajectories by a modified DWA method. Chang et al. [23] modify the DWA factors and use a tabular Q-learning method to choose an appropriate combination of weights in the cost function. However, their method requires a localization system in order to calculate the added component $osscillation$.

To address the problem of needing different weights for different scenarios, Hong et al. [24] propose the usage of a fuzzy system to adapt to the weights of the algorithm based on the distribution of obstacles around the robot. However, obstacle distribution is one of eight types, and the fuzzy rules are set by hand. Teso-Fz-Betoño et al. [25] expand this idea and introduce an artificial neuro-fuzzy inference system for predicting the weights and introduce the optimization of the cost function for two steps ahead. In the inputs to their neuro-fuzzy predictor, they only consider the mean of the obstacle distances, the heading, and the current speed, and not the detailed distribution of obstacles. We investigate the performance of this method (ANFIS DWA) in our evaluations.

In the past few years, numerous approaches that use a neural network to model the complete navigation policy have appeared [6], [7], [8], [10], using RL to train the network. Xie et al. [6] have developed a monocular image obstacle avoidance method based on the dueling and double-Q mechanisms. This method, however, does not navigate to a goal location. Zhu et al. [7] develop a method for navigating to a target given as an image in a previously known environment. Leiva et al. [26] learn a collision avoidance policy, which as input combines the data from the RGB, depth image, and lasers on the robot. Tai et al. [10]

train a navigation policy for the TurtleBot using deterministic policy gradients. Devo et al. [27] train separate exploration and localization networks for visual target-based navigation. Francis et al. [28] present a method RL-based navigation policy, which is also used to modify a probabilistic road map to the space that is reachable by the navigation policy.

Learning-based methods have also addressed the problem of navigating in human-occupied areas as well. Kretzschmar et al. [29] present a method that uses inverse RL to learn to imitate humans in their movement interactions to create a socially compliant navigation method. Everett et al. [30] learn policies of cooperation with other moving agents by simulating multiple agents and a strategy using long short-term memory to process an arbitrary number of other agents. Guldenring et al. [12] develop a method for training agents for human-aware indoor navigation (HAIN) by simulating the laser scan of the human leg motion. They use an RL method to train their policies in a few maps of real scanned spaces.

The issue of directly generating movement commands from a neural network is that we cannot guarantee collision-free trajectories for unseen inputs, which is especially important when a robot is moving near humans. The DADWA exploits the data modeling capabilities of neural networks, but the motor commands are generated with consideration to the accelerations of the robot, such that the robot can always stop in time. We start from our previous work in navigation in static environments [13] and create a new method for navigation in dynamic environments.

We train our agent in an RL setting with the PPO method [14]. PPO is a more efficient variant of the trust region policy optimization algorithm; however, the costly Kullback–Leibler divergence calculation has been replaced with a much computationally simple limitation on the permitted changes in action probabilities between updates. We use it because of its elegant and efficient implementation and demonstrated convergence properties.

III. DWA Considerations

A. DWA Method

In order to explain our approach, we will make a short introduction to the DWA. In the DWA, we assume that the control intervals are sufficiently small so that in each control interval $[t_i, t_{i+1}]$, we can consider the velocity of the robot to be approximately constant.

If we assume a differential-drive robot and if the translational velocity of the robot in the $[t_i, t_{i+1})$ interval is ν_i, its rotational velocity is ω_i, and its orientation at time t_i is $\theta(t_i)$, then we can derive that during the given interval, the robot will be moving along a circular arc with a center in

$$\left(x_i - \frac{\nu_i}{\omega_i}\sin\theta(t_i), y_i + \frac{\nu_i}{\omega_i}\cos\theta(t_i)\right) \tag{1}$$

and a radius of $r = \frac{\nu_i}{\omega_i}$ assuming $\omega_i \neq 0$ [1]. If $\omega_i = 0$, then the robot will move along a straight line in the direction $\theta(t_i)$.

Knowing this, at each control interval, the 2-D search space $[\nu, \omega]$ for the command is reduced to the intersection of the admissible velocities and the dynamic window. Admissible velocities are all velocities that will not cause a collision and, in other words, allow the robot to stop before reaching an obstacle on the current curvature, given its deceleration limitations. The dynamic window is the 2-D (translation and rotation) space of all velocities that can be reached within the next control cycle, given the limited acceleration and deceleration of the robot.

According to the DWA, from this search space, the best velocities for the next control cycle are the ones that maximize the function [1]

$$G(\nu, \omega) = \sigma(\alpha \cdot h(\nu, \omega) + \beta \cdot c(\nu, \omega) + \gamma \cdot v(\nu, \omega)) \tag{2}$$

where $h(\cdot)$, or *heading*, is the angle between the forward direction of the robot and the goal, a measure of how much the robot is moving toward the goal; $c(\cdot)$, or *clearance*, is a measure of the distance to the closest obstacle on the considered curvature; $v(\cdot)$, or *velocity*, is simply the translational velocity of the robot, so as to maximize its speed of movement; and $\sigma(\cdot)$ is a smoothing operator that normalizes $h(\nu, \omega)$, $c(\nu, \omega)$, and $v(\nu, \omega)$ to the range [0,1]. The different factors of the DWA are visualized in Fig. 4.

We note here that the behavior of the robot will change with different values of α, β, and γ. Our investigation of this dependence is presented in the next section. The optimal setting of these weights depends on the positions of the obstacles around the robot, and in this work, we design a neural network and a training scheme to predict these weights.

B. DWA Shortcomings

In practice, we have to answer several key questions about the implementation before using the DWA. The *velocity* component is straightforward and can be calculated directly. The *heading* component, in the original paper, is calculated at the position the robot would reach after applying maximum deceleration. It is not immediately clear why this is the case, and in some subsequent popular implementations, it is at the robot position in the next control cycle [2]. The most problematic is the *clearance* component for which many "edge" cases exist. For this component, we need to answer the following questions.

1) If the command is causing the robot to rotate in place, and there are no obstacles in its circumscribed circle, what should the value of the *clearance* be? The robot does not encounter any obstacles, but it also does not approach the goal.
2) What should the value of *clearance* be if there is an obstacle in the circumscribed circle? It seems that the clearance should be defined in terms of angular distance, but this will not be comparable to other commands.
3) What about commands where the robot would be moving along a circle with a circumference smaller than the maximum value for the clearance? Should the value of the clearance component be the length of the circumference or the maximum clearance value?

Different answers to the raised questions will result in different behaviors of the robot, as illustrated in Fig. 2(d)–(f).

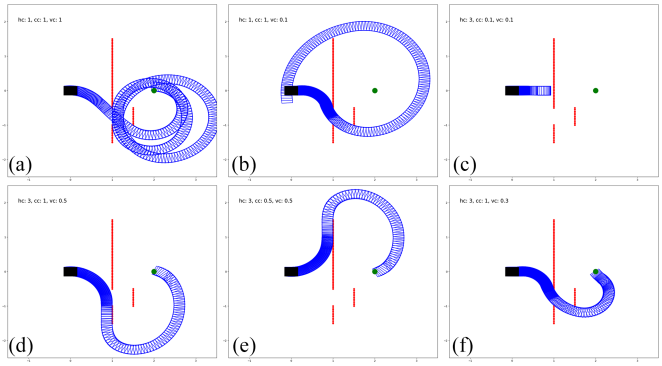

Fig. 2. Effect of different weights in the cost function for the three factors of DWA. The robot, starting at the black square, is trying to reach the green goal, while avoiding the red obstacles. In (a), the robot reaches the goal but moves along an extremely suboptimal trajectory. In (b) and (c), the robot fails to reach the goal in the allotted time steps. In (e) and (d), the robot reaches the goal, but takes a long trajectory, not going through the gap in the obstacles. In (f), the robot reaches the goal by passing through the opening in the obstacles.

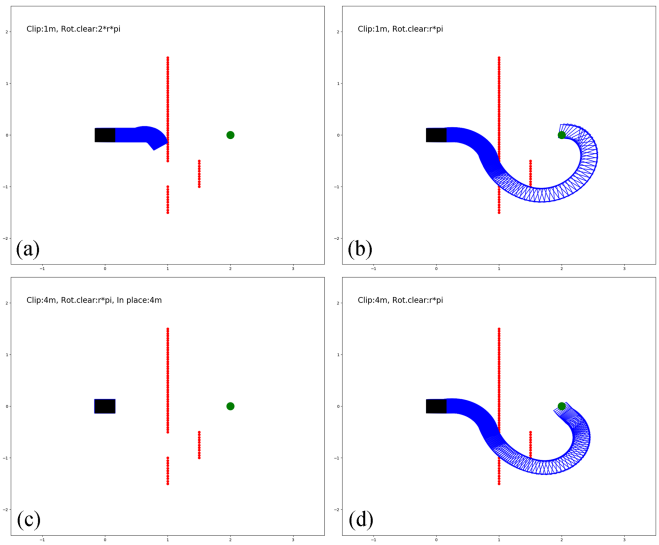

Fig. 3. Effect of different design choices in the implementation of the DWA. The robot, starting at the black square, is trying to reach the green goal, while avoiding the red obstacles. In (a), the *clearance* factor is clipped at 1 m, and the *clearance* in the case of rotation without obstacles is set to the length of the curvature. In (b), the *clearance* in the case of rotation without obstacles is set to half of the length of the curvature. In (c), the *clearance* is clipped at 4 m and is set to 4 m in the case of an in-place rotation. In (d), the *clearance* is clipped at 4 m, and in the case of rotation without obstacles, it is set to half of the length of the curvature.

Next, we must also normalize the factors, so that they are comparable in the cost function. For this, it is necessary to set limits to the maximum values of all factors, and this clipping limit is a design decision that, again, will greatly impact the behavior of the robot, as illustrated in Fig. 3(a)–(d). The original paper also reduces the weight of the velocity factor when the robot is near obstacles, which is another hyperparameter that needs to be set.

Finally, we need to decide on the weights of the different factors. However, the optimal weights for the factors in the cost function cannot be defined for all navigation scenarios, as they depend on the distribution of obstacles around the robot, as well as the details of the normalization of each factor in the cost function. Some immense effects of the changes in these weights are also illustrated in Fig. 2.

TABLE I
BEST PARAMETERS FOR DWA PER DIFFERENT ENVIRONMENT IN THE IN2D DATASET

Environment	Best parameters			Environment	Best parameters		
	α	β	γ		α	β	γ
1	0.8	0.2	0.1	10	0.7	0.4	0.1
2	1.0	0.1	0.2	11	0.8	0.3	0.1
3	0.8	0.5	0.1	12	0.8	0.1	0.5
4	0.9	0.1	0.5	13	0.5	0.4	0.1
5	0.8	0.1	0.1	14	1.0	0.1	0.3
6	0.7	0.3	0.1	15	0.8	0.1	0.1
7	1.0	0.2	0.3	16	0.6	0.3	0.1
8	1.0	0.1	0.5	17	0.8	0.1	0.1
9	0.9	0.1	0.4	**Best overall**	0.8	0.1	0.1

The environments are visualized in Fig. 9.

TABLE II
BEST PARAMETERS FOR DWA, EVALUATED ON 170 EPISODES WITH EACH CONSISTING OF MULTIPLE STATIC INTERMEDIATE GOALS

α	β	γ	Number of completed episodes	Number of reached goals
0.8	0.1	0.1	66	511
1.0	0.1	0.3	60	488
1.0	0.1	0.5	52	403
1.0	0.1	0.1	50	435
0.8	0.5	0.1	50	392

To examine the sensitivity of DWA to the weights in the cost function, we performed a grid search of the α, β, and γ weights, evaluating the performance of the algorithm with the different weights on the complete IN2D dataset [13], [31], visualized in Fig. 9. We evaluated all the combinations of values for the weights, in the range [0,1] with a step of 0.1. The range is limited since the best command is chosen as the one that produces a maximum of the cost function, which means that only the relative scaling of the factors is relevant. The evaluation was performed for ten different navigation tasks for each of the 17 environments, with each navigation episode consisting of randomly sampled start and goal locations, as well as intermediate goals set at a distance of 1 m on the shortest path between the two locations.

In Table I, we show the best performing combination of the weights for each of the 17 rooms in the dataset. From the results, it is evident that different weights for the cost function are required in order to achieve the best performance for each environment. In Table II, we show the five best performing weights in the cost function overall. The results in both tables confirm that the DWA is sensitive to the weights in the cost function. We also conclude that in order to get acceptable behaviors, α should be about two to ten times larger than β and γ. In the instances where this was not the case, the algorithm usually underperformed. These results show that the performance clearly depends on these parameters.

For practical applications of the algorithm, different implementations are used, due to different decisions regarding the raised questions, as well as other details (grid-based evaluation versus working in continuous space). For example, the implementation in the popular ROS package *dwa_local_planner* changes the heading factor in the cost function to the distance

to the goal, moves the goal along the global path at each cycle, and introduces an additional factor—distance to the global path. While this produces good results, it also makes it impossible for the planner to be used in the considered setting in this article, without a map and a global planner.

IV. Dynamic Adaptive Dynamic Window Approach

A. Navigation as a Markov Decision Process

In this article, we propose a method to predict the optimal values of the weights of the DWA cost function based on the obstacles around the robot. To solve the problem with an RL method, we formulate it as a Markov decision process:

1) State: The state $s \in S$ (or observation, under the assumption of a fully observable environment) is defined as the distances to obstacles around the robot, the relative location of the goal, and the dynamic parameters of the robot

$$s_t = \left(l_t^t, l_t^{t-3}, l_t^{t-7}, d_t, \phi_t, \nu_t, \omega_t\right) \tag{3}$$

where $l_t^{t-\tau}$ is a distance measurement of N distances around the robot $l_t^{t-\tau} = [l_1, l_2, \ldots, l_N]_t^{t-\tau}$ taken at τ control cycles in the past and represented in the robot frame at the current location, d_t and ϕ_t are the distance and angle to the goal, while ν_t and ω_t are the current translational and rotational velocities of the robot.

2) Action: At each step, the agent receives an observation (or state) of the environment and outputs an action $a \in A$. We define an action as setting the weights of the cost function for the next control interval, which will be used for generating the movement command for the robot

$$a_t = (\alpha_t, \beta_t, \gamma_t, \delta_t) \tag{4}$$

where α_t, β_t, and γ_t are the weights for the original DWA cost function that will be used for calculating the next movement command and δ_t is an additional weight for the extended cost function, explained in the next section.

3) Transition Function: The transition from one state to another is implicitly determined by the dynamics of the robot, its perception capabilities, and the simulation environment in which the robot operates (and the network is trained)

$$T(s, a) : S \times A \to S \tag{5}$$

since we are using a model-free RL method for training the network, there is no need to explicitly model this function.

4) Reward Function: In general, the reward function is defined as a "large" positive value for a desirable state (reaching the goal) and a "large" negative reward for being in an undesirable state, like crashing or being stuck in one place

$$R(s_t) = r_t, \quad r_t \in \mathbb{R} \tag{6}$$

or more specifically, at time t, the robot receives a reward of

$$r_t = \begin{cases} R4, & \text{if collision}_{\text{while robot is still}} \\ R3, & \text{if collision}_{\text{while robot is moving}} \\ R2, & \text{if stuck} \\ R1, & \text{if goal reached.} \end{cases} \tag{7}$$

$R1$ is the reward for reaching the goal, $R2$ is the negative reward for not moving multiple control cycles, $R3$ is the negative reward for colliding with a human, and $R4$ is a very small negative reward for colliding with a human, while the robot is not moving (or moving with a negligible velocity). The robot receives a neutral reward for a collision, while it is still because we interpret this as not the fault of the robot since it cannot escape the fast-moving pedestrians.

5) Discount Factor: The discount factor, which defines how much less we care about the reward in the next step instead of the reward at the current step, is set to

$$\gamma_{\text{discount}} = 0.99. \tag{8}$$

With the previous values defined, our goal becomes finding the optimal parameters of a parameterized policy (mapping from states to actions). Since the (stochastic) policy π_w is represented with a neural network and modeled as a diagonal Gaussian policy

$$a_t \sim \pi_w(\cdot|s_t) \tag{9}$$

we are optimizing the network weights w so as to find the optimal policy, defined as the policy that maximizes the expected cumulative reward of an episode

$$w* = \underset{w}{\text{argmax}} \underset{\pi_w}{\mathbb{E}} [J(\rho)]. \tag{10}$$

The expected cumulative reward of an episode of T steps

$$\underset{\pi_w}{\mathbb{E}} [J(\rho)] = \underset{\pi_w}{\mathbb{E}} \left[r_0 + \gamma_{\text{dis}} r_1, \ldots, \gamma_{\text{dis}}^{n_{\text{steps}}} r_{n_{\text{steps}}}\right] \tag{11}$$

depends on the policy that generates the actions, the start-state distribution, the transition function, as well as the reward function. Since the policy generates the actions for the observed states, it is also referred to as the actor.

The algorithm we use for training the network is the actor–critic RL method PPO [14], which is an actor–critic RL method that directly optimizes the policy, but most crucially has an efficiently computable limitation to the change in the policy between successive updates, resulting in more stable training. The policy is a stochastic continuous policy, and the output is a diagonal multivariate Gaussian distribution, while the standard deviations are fixed as hyperparameters.

In PPO, the critic is a parameterized value function $V^{\pi_w}(s)$, which approximates the expected cumulative return after being in state s

$$V^{\pi_w}(s) = \underset{\pi_w, s}{\mathbb{E}} \left[r_0 + \gamma_{\text{dis}} r_1, \ldots, \gamma_{\text{dis}}^{n_{\text{steps}}} r_{n_{\text{steps}}}\right] \tag{12}$$

which can also be referred to as the *value* of a state. The value function is also modeled by a neural network, and in this work, it shares the structure of the actor network, with the exception of the last layer, and it is trained by backtracking the rewards in an episode.

During training, we use the critic network to calculate the advantage function $A^{\pi_w}(s, a)$, which informs how much more reward is expected in state s after taking action a, rather than the "average" action. Its purpose is the reduction of the variance, which is a common issue in RL methods.

For the training of the actor, the advantage function is calculated as

$$A^{\pi_w}(s_t, a_t) = r_t + \gamma_{\text{dis}} r_{t+1} + \cdots + \\ + \gamma_{\text{dis}}^{t+n_{\text{steps}}} r_{t+n_{\text{steps}}} - V^{\pi_w}(s_t) \tag{13}$$

that is we calculate the difference between the total return after taking action a_t in state s_t and the estimated "value" of state s_t.

Finally, we can define the objective function that is used for updating the policy

$$L(s, a, w_k, w) \\ = \min\left(\frac{\pi_w}{\pi_{w_k}} A^{\pi_w}(s,a), \text{clip}\left(\frac{\pi_w}{\pi_{w_k}}, 1-\eta, 1+\eta\right) A^{\pi_w}(s,a)\right) \tag{14}$$

where π_{w_k} is the policy that generated the experience and η is a constant for clipping the change in action probabilities that will happen after updating the network weights. The weights of the network are updated to maximize the objective function

$$w_{k+1} = \underset{w}{\text{argmax}} \underset{\pi_{w_k}}{\text{E}} [L(s, a, w_k, w)]. \tag{15}$$

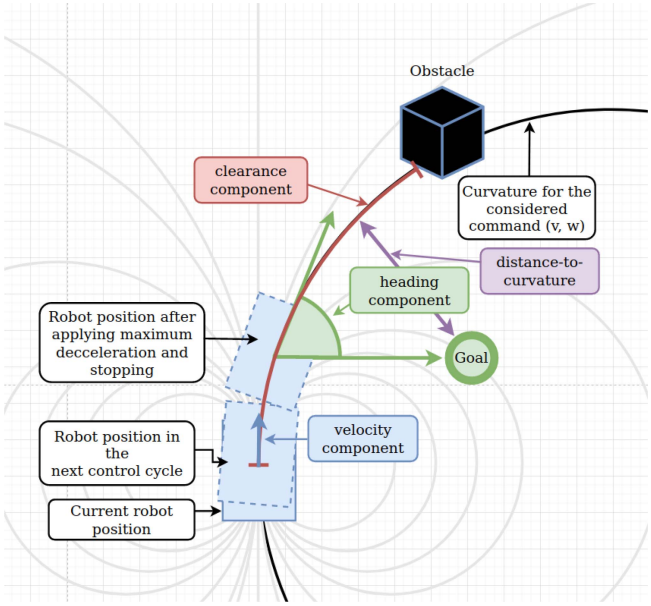

Fig. 4. Representation of the different components of the (extended) DWA. The gray lines represent the resulting robot movement curvatures for different commands from the dynamic window. The blue, red, and green components represent the three factors in the original cost function, while the purple factor is added in this work.

B. Distance-to-Curvature Component

We observed an issue that exists in both the DWA and the ADWA when the goal position is not near the robot. Namely, when the robot would stop or almost stop near an obstacle, it would enter oscillatory mode and would fail to start moving again. Our experiments indicate that the main cause of this behavior was the *heading* component in the cost function, as in these situations the robot must choose a command with a lower value for this component to reach the goal. On the other hand, the *heading* component has to have a relatively high weight to get the robot to the goal, and in situations where there is an obstacle in front of the robot and the robot is approximately still, the robot needs to somewhat temporarily disregard the heading component until it has nearly circumvented the obstacle.

To deal with this issue, we propose to include another component to the cost function, the *distance-to-curvature component*, as visualized in Fig. 4. The distance-to-curvature component measures the distance from the goal to the nearest point of the considered curvature. It is normalized as $\frac{(max\_dist - min(max\_dist, distance\_to\_goal))}{max\_dist}$, where $distance\_to\_goal$ is the measured distance and max_dist is the truncating factor as used for the $clearance$ component. The distance-to-curvature component has a value of 1 if the goal is on the curvature and a value of 0 if the goal is max_dist or more meters away from the curvature.

This factor has the function of giving the robot a "behind the obstacle" view of the goal and can temporarily "convince" the robot to move away from the obstacle. The complete cost function of our implementation of the DWA becomes

$$G(\nu, \omega) = \alpha \cdot h(\nu, \omega) + \beta \cdot c(\nu, \omega) + \gamma \cdot v(\nu, \omega) + \delta \cdot d(\nu, \omega) \tag{16}$$

where δ is the added weight for the new $d(\cdot, \cdot)$ component, which is the *distance-to-curvature* component. In our experiments, completely replacing the *heading* component for the *distance-to-curvature* component did not produce significantly better performance, as adhering strictly to curvature generally means sometimes choosing a slower speed, to follow the curvature. However, our experiments show that letting a neural network assign weights to both components depending on the observations does result in better performance.

C. History of Recent Observations

There are basically two ways of dealing with dynamic obstacles in mobile robotics. If the velocity of the robot is significantly higher than the velocity of the obstacles, we can simply include necessary tolerances in the movement commands and regard the obstacles as static. On the other hand, if the obstacles move with approximately the same or higher velocity than the robot, it is necessary to consider their future positions.

In the considered environments, the robots do not move faster than the humans, so our approach is to endow the neural network with some knowledge of where the obstacles might be in the near future. To do this, we consider the state to be a recent history of aligned observations and let the network implicitly learn to model these positions during training.

Our goal is to let the neural network implicitly predict the future positions of the obstacles, by including observations of the obstacles at three different time steps in the past. To do so, the main input of the network is three observations stacked in a 1-D three-channel input. Each channel is a laser scan observation of the environment consisting of N range scan readings around

the robot, represented as a simple vector of distances $l_t \in \mathbb{R}^N$ at a fixed angle for each element (determined by the design of the laser scanner). The first channel is the current observation of obstacles around the robot. The second channel is the observation of three control cycles in the past, and the third channel is the observation of seven control cycles in the past. Crucially, the past laser scans are transformed into the frame of the current robot position

$$l_t^{t-3} = \text{pol}\left(T_t^{t-3} \cdot \text{cart}\left(l_{t-3}^{t-3}\right)\right)$$
$$l_t^{t-7} = \text{pol}\left(T_t^{t-7} \cdot \text{cart}\left(l_{t-7}^{t-7}\right)\right) \quad (17)$$

where $l_t^{t-\tau}$ is a distance measurement of N distances around the robot taken at τ control cycles in the past and represented in the current robot frame, pol$(\cdot)$ is a function that transforms points in Cartesian coordinates to a range scan (quasi polar coordinates), $T_t^{t-\tau}$ is the transformation from the robot frame at time $t-\tau$ to the robot frame at time t, and cart$(\cdot)$ is a function that transforms a range scan to points in Cartesian coordinates. Put in words, the past laser scans are transformed to Cartesian coordinates, then transformed in the frame of the current robot position, and then, by using binning and interpolating the eventual unknowable values, transformed as they would have been seen from the current robot position. These observations are clipped at a certain fixed distance d_{clip}.

D. Learning to Predict the Weights

The optimal weights for the DWA depend on the distribution of obstacles around the robot, the current velocity of the robot, and the position of the goal. Since range scans are highly correlated spatially, in the design of the network, we use convolutional layers to process the stack of recent observations. After a few convolutional layers, they are processed by a single fully connected layer whose output is concatenated with the relative position of the goal and the current velocities of the robot. The joined data are then passed through several fully connected layers before finally outputting the weights to maximize the expected future reward of the robot. The detailed design of the network is shown in Fig. 5. The outputs of the network are linear output neurons, which represent the values of a diagonal Gaussian distribution, one output for the mean value of each weight in the extended DWA cost function.

The network represented in Fig. 5 represents the actor, while the critic has the same network design but with a single linear output predicting the expected total return of the state/observation, or how much reward, on average, the agent can expect to get until the episode is terminated, given that at time t, the agent is in state s_t.

The training is done episodically and in two stages. At the start of each episode, the robot is spawned at a random location on the map. A reachable goal is randomly chosen within a fixed distance $d_{\max}$ from the robot's starting location. The episode ends when the robot reaches the goal and has exceeded the maximum of n_{steps} time steps or when a human has walked into the robot.

In the *first stage* of training, the robot is navigating without any moving obstacles. We found that in the beginning, the robot is moving almost randomly, and in the presence of moving obstacles, it rarely reaches the goal, and the learning fails. For this reason, in the first stage, we intend to get the robot to move to the goal most of the time, so that it can continue to learn in the presence of moving obstacles.

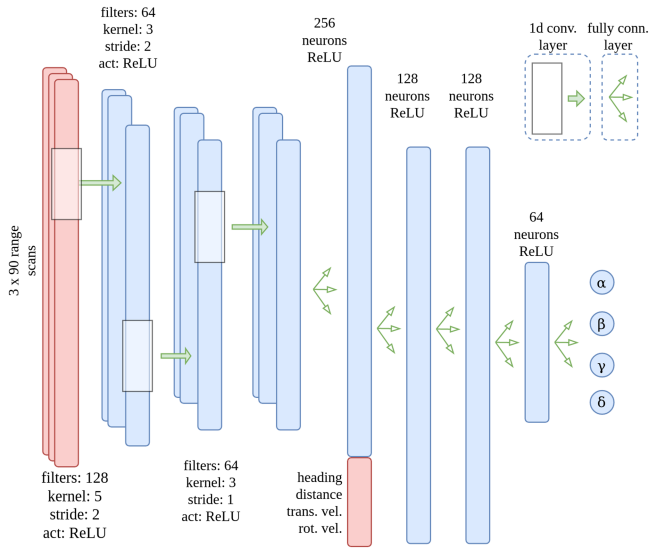

Fig. 5. Input of the network is three stacks of laser range data as 1-D three-channel input. These are passed through three convolutional layers and then unrolled into a single vector. This vector is concatenated with the velocities of the robot and the heading and distance to the goal and are then passed through four additional fully connected layers. The outputs of the network are the four weights of the DWA.

In the *second stage* of training, the robot learns to navigate in the presence of humans. During each episode, pedestrians are spawned near the robot, and the robot learns and relearns to navigate to the goal. The training for both stages is done in four simulations in rather different environments: two of real scanned spaces, one virtually designed space, and one large virtual obstacle field, which are run in parallel. The training environments are visualized in Fig. 6. The observations, actions, and rewards gathered by the four agents are collected in a common buffer and used to update the single network that controls the agents.

During training, the robot receives a reward at each time step, and the reward structure that we use is composed of several components. First, the robot receives a large positive reward of $R1$ if it reaches the local goal. Second, it receives a negative reward $R4$ if it gets stuck in the same position for n time steps. Finally, it receives a large negative reward $R3$ if a collision with a pedestrian happens while the robot is moving. If a collision happens while the robot is still, it is not punished.

V. Experiments

A. Simulation Details

For the base simulation, we are using the Flatland[1] simulator, which is a 2-D robotics simulator that can run significantly faster than more comprehensive simulators such as Gazebo[2]

[1] Avidbots, Flatland. [Online]. Available: https://github.com/avidbots/flatland
[2] Gazebo simulator. [Online]. Available: http://gazebosim.org/

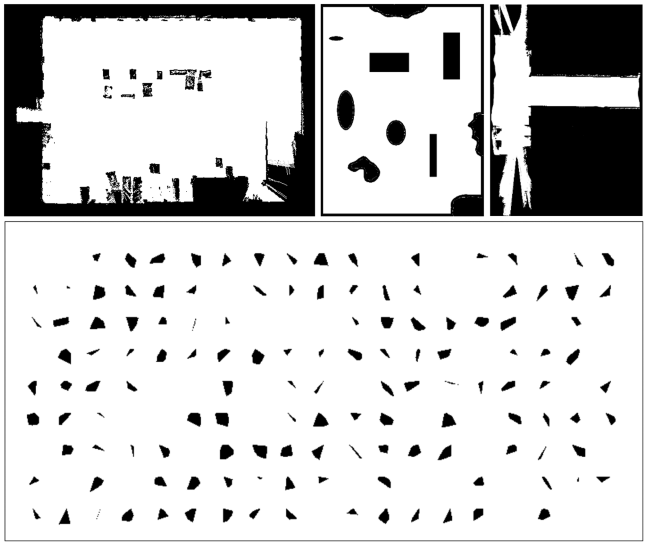

Fig. 6. Four environments used for training the robot. Two real scanned spaces, one environment with simple geometric obstacles, and a large hall with randomly generated convex obstacles.

or CoppeliaSim.[3] The simulator integrates seamlessly with the ROS and can simulate laser range scanners with added noise in the measurements. We use two separate virtual laser range finders: one that detects the obstacles and one that detects the humans. These laser scans are then joined with a *min* function to form a single realistic observation. In addition, since the simulator updates the velocity instantaneously, we use a separate node that preprocesses the commands that come out from the DWA before updating the robot in simulation, simulating the limited acceleration and deceleration of the real robot.

The pedestrians are simulated using the PedSim[4] pedestrian simulator. This library uses Helbing's social forces model [32] to simulate the movement of pedestrians among static obstacles and other pedestrians. This model simulates multiple potential fields from the obstacles, other pedestrians, and the goal destination. In addition, we also use a plug-in for simulating the gate of each pedestrian [12], with some randomness in the size of the gait, legs, and speed of movements of the pedestrians. This is done to simulate an accurate sensor reading, considering the height of the sensor of our robot.

B. Experimental Setup

In our experiments, the goal location is randomly generated at a distance less than or equal to $d_{\max} \in [1,8]$ m from the starting location. After the A* planner is run, intermediate goals are chosen at a distance of $d_{\text{inter}} = 1.5$ m apart. The number of range scan readings at each point in time is $N = 90$, and all measurements are clipped at $d_{\text{clip}} = 4$m. Each episode is run for a maximum of $n_{\text{steps}} = 150$, after which it is considered unsuccessful and it is terminated. The reward $R3$ is -50 if the robot collides with a pedestrian while the robot moving, if it collides while the robot is still, the episode is terminated, but no negative reward is given to the agent ($R4 = 0$). When the robot reaches the goal, it receives the reward $R1 = 100$, and if it does not move for 20 consecutive control cycles, the negative reward $R2$ with a value of -20 is incurred. During the first stage, the training is done for $3M$ time steps, after which the second stage of training is run for $30M$ time steps.

We are using a model of the ATRVmini robot for which we identified the maximum acceleration of the robot to be 0.43 m/s^2 for the translational component and 1.4 rad/s^2 for the rotational component. The maximum allowable translational velocity of the robot is set to 0.7 m/s.

During the *second stage* of training, the humans that are added to the simulation are set not to avoid the robot. We found that if the simulated humans are set to avoid the robot, the policy network did not learn to avoid the humans, as there was no need for this.

C. Evaluation Setup

The evaluation is performed on the IN2D dataset [13], which contains 2-D maps of scanned home and office spaces, which was generated using a subset of the Gibson dataset [31], which contains 3-D data of scanned indoor spaces. All 17 environments are depicted in Fig. 9. If the space contained multiple floors, we generated a separate 2-D map for each floor. In each space, we generated both single-goal and multiple-goal navigation episodes. Each episode was evaluated with and without walking humans in the space.

To isolate the effects of the local planner on the success of navigation, we thus simplify the interaction between the local and the global planner as much as possible. The global planner is run only to create navigation scenarios and does not interact with the evaluated navigation methods in any online way. This also results in some goals being placed in places where pure local navigation cannot reach the goal effectively (maze-like tasks). In production, a navigation system would include a more complex interaction between the global and the local planner, such as taking care of where the targets for the local planner were placed given the velocity, position, and orientation of the robot, which would include recovery behaviors and other interactions. However, in this evaluation, we focus only on the local planner, since the improvement in the performance of this component naturally translates in the improvement of the overall performance as well.

In the evaluation scenarios containing humans, the human agents are set to be repelled by the robot agent with a very small random force. In tight corridors and doorways, it is impossible for the robot to pass if there are a lot of people crossing or standing in the way, completely ignoring the robot. It is more realistic for humans to react in some way to the robot. However, despite this small force, the simulated human agents walk into the robot, only occasionally deciding to avoid it. The simulated agents are very much more aggressive than real humans.

[3]CoppeliaSim (formerly V-REP). [Online]. Available: https://www.coppeliarobotics.com/

[4]Pedestrian simulator. [Online]. Available: https://github.com/srl-freiburg/pedsim_ros

TABLE III
FIRST EXPERIMENT: STATIC ENVIRONMENT, SINGLE DISTANT GOAL, AND EVALUATION ON THE IN2D DATASET

Method	Number of successes	Number of failures	Number of collisions	Number of timeouts
DWA [1]	980	720	**0**	720
ANFIS DWA [25]	992	708	**0**	708
ADWA [13]	1223	477	**0**	477
HAIN [12]	1305	395	216	**179**
DADWA	**1467**	**233**	**0**	233

The best values are in bold.

TABLE IV
SECOND EXPERIMENT: DYNAMIC ENVIRONMENT, SINGLE DISTANT GOAL, AND EVALUATION ON THE IN2D DATASET

Method	Number of successes	Number of failures	Number of collisions	Number of timeouts
DWA [1]	722	978	358	620
ANFIS DWA [25]	698	1002	325	677
ADWA [13]	802	898	413	485
HAIN [12]	1157	543	365	**178**
DADWA	**1293**	**407**	**209**	198

The best values are in bold.

D. Results

In the evaluation scenario, we evaluate the methods on the ATRVmini robot, depicted in Fig. 15. We evaluate DADWA, DWA [1], ANFIS DWA [25], ADWA [13], and HAIN [12]. In all experiments, the methods were evaluated in terms of the number of successfully completed episodes and the number of failed episodes. The number of failures is separated into the number of failures due to collisions and the number of failures due to timeouts—times the robot did not collide or reach the goal in 300 control cycles.

In the *first experiment*, we evaluate each method on a total of 1700 single goal navigation tasks, with 100 episodes consisting of a single distant goal for each of the 17 environments used in the experiment. The evaluation was performed in static environments, without any moving obstacles. The results of this evaluation can be seen in Table III. From the results, it is evident that the methods that are based on the DWA do not cause collisions with the environment, since the DWA by design does not produce commands that would result in collisions. The DWA gets stuck sometimes, as the distance to the goal is relatively large and the goal is often in a position that cannot be reached by looking ahead only a single control interval. In practice, this problem is solved by implementing a recovery behavior and reducing the distance to the given goal. The neural-network-based HAIN algorithm did cause collisions with the environment.

In the *second experiment*, the robots have the same navigation tasks, but now moving pedestrians are added near the ideal path of the robot, as represented in Fig. 8(b). In Fig. 10, we can see an example of a human crossing the path of the robot. We can see that the robot swerves to the right as the human approaches. As the human passes and is no longer a threat, the robot returns to its movement toward the goal. In Fig. 11, we see an example of the robot making a tight crossing with a human near a passage to a different room. As the robot approaches the human, it turns to the right after which it successfully enters the partially blocked room. As expected, from the results in Table IV, we can see that as the pedestrians are added to the paths, all methods have

Fig. 7. ATRVmini robot used in the experiments. A model of the robot was used in the simulation, and the policy was tested on the real robot. The robot is equipped with a laser range scanner and localization beacons, among other equipment.

TABLE V
THIRD EXPERIMENT: STATIC ENVIRONMENT LONG EPISODES AND EVALUATION ON THE IN2D DATASET

Method	Number of successes	Number of failures	Number of collisions	Number of timeouts
DWA [1]	882	818	**0**	818
ANFIS DWA [25]	890	810	**0**	810
ADWA [13]	887	813	**0**	813
HAIN [12]	1012	688	245	**443**
DADWA	**1197**	**503**	**0**	503

The best values are in bold.

failures due to collisions. The increased number of collisions is due to the pedestrians colliding with the robot or the robot colliding with the pedestrians. It is nontrivial to differentiate between the two. Given the acceleration limits of the robot and the distribution of movement speeds of the pedestrians, the robot is unable to avoid pedestrians moving directly into the robot, as shown in Figs. 12 and 13.

In the *third experiment*, we evaluate each method on a total of 1700 multiple goal navigation tasks in static environments, with 100 episodes consisting of multiple intermediate goals placed at a distance of 1.5 m on the ideal path to the final goal. The results are presented in Table V . We can see that as the length of the navigation episodes increased, the failures also increased, which is to be expected as the chances for failure also increased. Interestingly, the increase in failures is not proportional to the increase in path length, as the complete paths were on average three times the length of the single-goal tasks. We believe that this is due to the failures in DWA being much more probable when the robot is standing still or moving at low velocities.

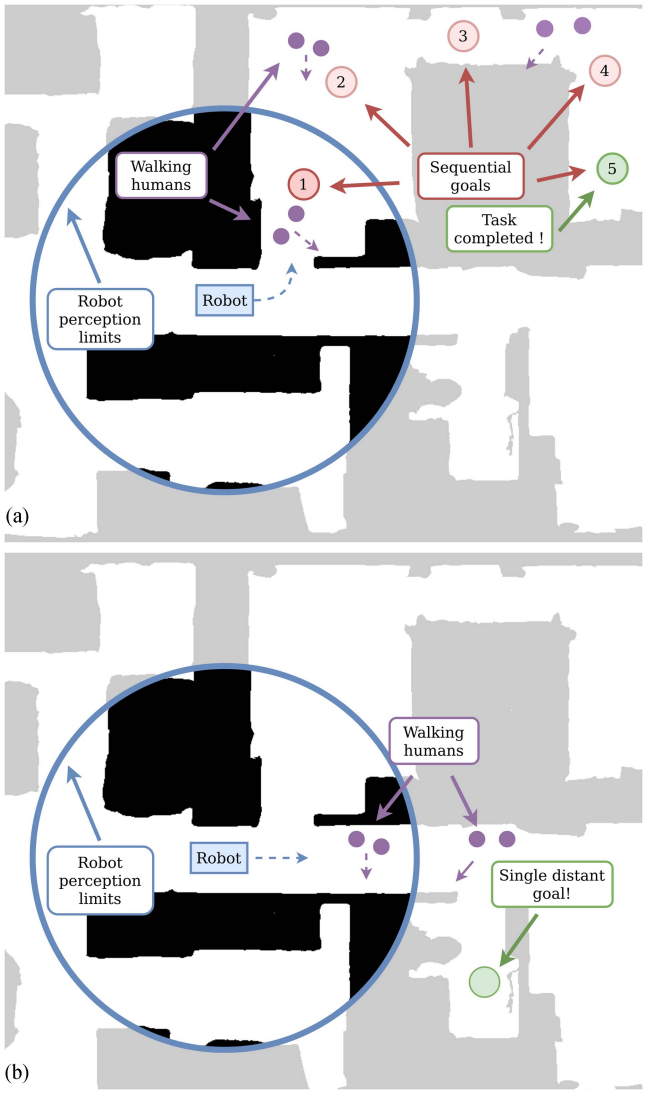

Fig. 8. (a) Long episode experiments. The robot knows only the next goal, and after reaching it, it is supposed to navigate to the next goal. The episode is successfully completed when the final goal is reached. (b) Second type of experiment when a single distant goal is provided to the robot. Both settings become dynamic when the visualized pedestrians are added to the simulation.

TABLE VI
FOURTH EXPERIMENT: DYNAMIC ENVIRONMENT LONG EPISODES AND EVALUATION ON THE IN2D DATASET

Method	Number of successes	Number of failures	Number of collisions	Number of timeouts
DWA [1]	611	1089	423	666
ANFIS DWA [25]	595	1105	703	**402**
ADWA [13]	620	1080	647	433
HAIN [12]	851	849	377	472
DADWA	**973**	**727**	**298**	429

The best values are in bold.

TABLE VII
DYNAMIC ENVIRONMENT SHORT EPISODES AND EVALUATION ON THE IN2D DATASET

Method	Number of successes	Number of failures	Number of collisions	Number of timeouts
DADWA 1RS	**1078**	622	419	203
DADWA 3RS	**1293**	407	209	198

The best values are in bold.

Since the goals are determined by a path planner, once the robot reaches one goal, it is often facing the next goal and moving with a larger velocity. On the other hand, when the robot starts navigating toward the first goal, it is standing still and facing in a random direction.

In the *fourth experiment* (see Table VI), the episodes are kept the same as in the third experiment, but moving humans are spawned near the path of the robot, as represented in Fig. 8(a). As can be expected, the performance drops for all methods. However, we can see that again our method completed significantly more episodes as compared to the rest of the evaluated methods.

In the first to fourth experiments, we test the usability of our local navigation approach as a stand-alone navigation system for long episodes in complex environments. In such a scenario, local navigation approaches cannot navigate to all the goals, as sometimes the goals are in places that are unreachable by purely local navigation methods. For the most part, this explains the failures of the methods in static environments. To demonstrate the usage of our method as a subsystem of a fully fledged navigation system, which has a map of the environment and uses a global planner, we conduct *the fifth experiment*. We take the navigation episodes from the third experiment and generate intermediate goals at 0.2 m, 0.5 m, and 1 m apart. The results are presented in Fig. 14.

From the results, it is evident that as the distance to the local navigation goal is reduced, both the basic DWA and our approach can successfully cope with the navigation task. As the distance to the local goal is increased, both approaches show increasing failures, with DADWA significantly outperforming the DWA.

Finally, to illustrate the significance of past observations for navigation in dynamic environments, we trained the same network with only a single laser scan (1RS) as the input. From the results presented in Table VII, we can clearly see that the dynamic component in the observations helps to include the pedestrian information in the output.

E. Transferring to a Real Robot

To demonstrate that a policy trained in this way can be used on a real robot, we trained a navigation policy for our real robot. The robot has an ATRVmini chassis (see Fig. 7) and is equipped with a Hokuyo laser range scanner.[5] We performed identification of the acceleration profile of the robot and adapted the simulation to match these dynamic parameters, and adapted the simulated laser scanner to match the real one, which measures distances in a range of $\pm 120^{\circ}$. We used four localization beacons on the robot in order to update the relative goal position at each time step.

In the initial experiment, we designed a compact obstacle course within our laboratory, creating three distinct navigation challenges, each comprising multiple goals. The paths taken by the robot, along with the intermediate objectives, are depicted in Fig. 15. It is noteworthy that the policy, entirely developed

[5]Hokuyo laser range scanner. [Online]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=165

Fig. 9. Visualizations of all 17 environments in the IN2D dataset used for the evaluation of the navigation methods. The environments are real floor plans of 3-D scanned indoor spaces and contain realistic configurations of various obstacles. These maps were loaded in the ROS, and a simulation of the ATRVmini robot was navigated on each map.

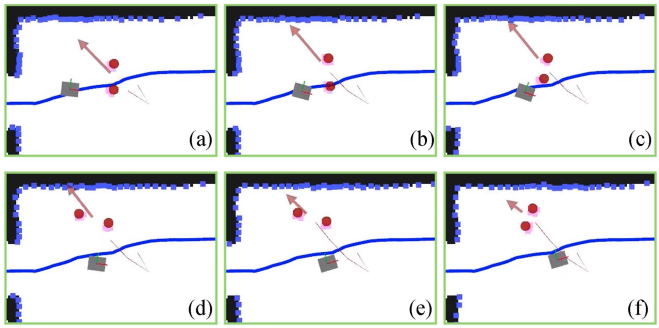

Fig. 10. (a)–(f) Interaction when the human is approaching from the right. The robot maneuvers to avoid it and then returns to the path to the next goal.

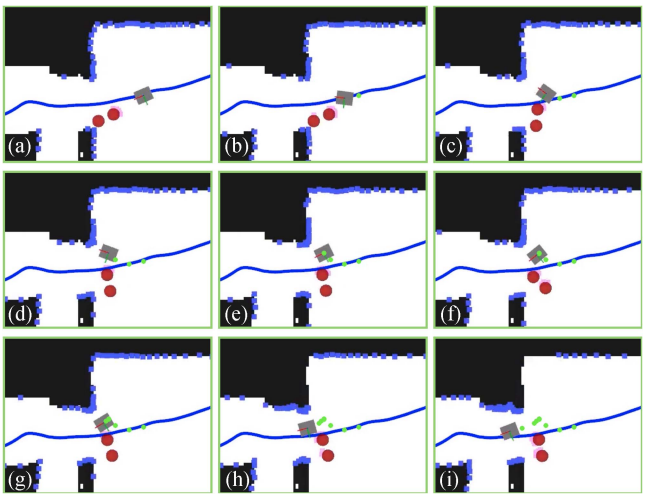

Fig. 11. (a)–(i) Example of a tight crossing into a closed space near the human. The robot hesitates but manages to cross safely.

through simulation, enabled a real robot to successfully complete the course.

In a subsequent experiment, we focused on guiding the robot to a singular far-off target. For this purpose, the robot was

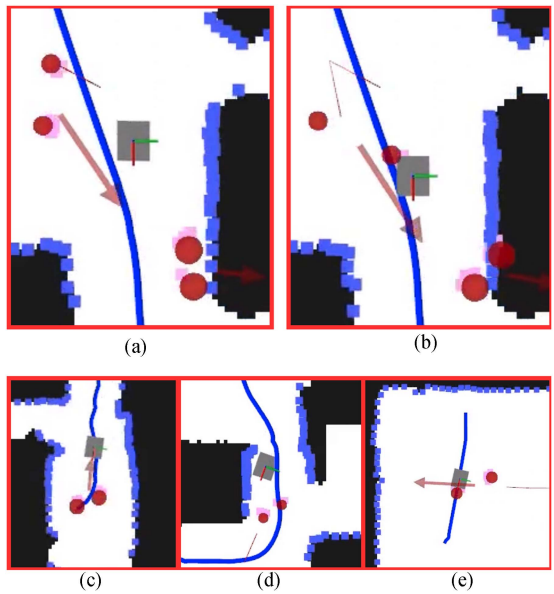

Fig. 12. Failures to reach the goal for our method. (a) and (b) Pedestrian approaching with great velocity from behind the robot. The robot cannot evade it due to its lower velocity and inability to evade. (c) and (d) Situations when there was no collision, but the robot did get in a stalemate position with the pedestrian. (e) Fast pedestrian moving into the robot from the right side.

outfitted with a RealSense D415[6] camera, mounted atop a pan-tilt mechanism. The CSR-DCF [33] visual tracking algorithm was run on the RGB image from the camera, and we used the depth data to calculate its position in the robot frame . We designated a static target. The path it followed is illustrated in Fig. 16. This experiment underscored the adaptability of the simulation-trained policy in achieving long-distance navigation tasks.

In the third and fourth trials, the objective shifted to pursuing a moving human target while circumventing obstacles, employing

[6]RealSense D415. [Online]. Available: https://www.intelrealsense.com/depth-camera-d415/

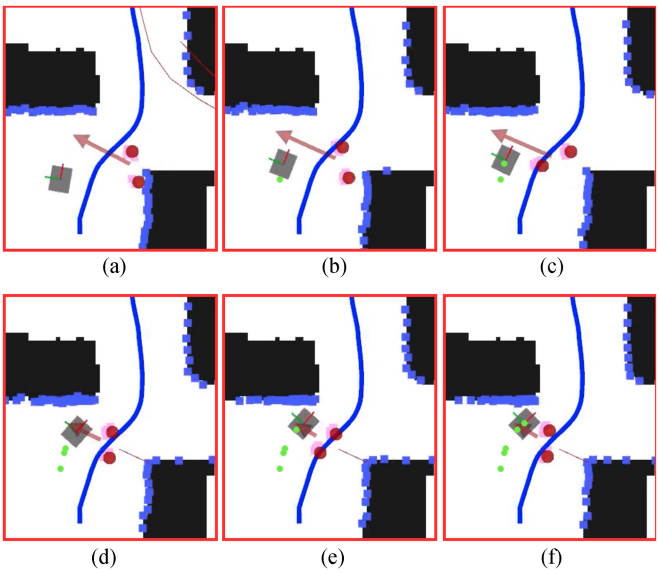

Fig. 13. (a)–(f) Development of the robot and the pedestrian blocking each other. The episode ended in a timeout.

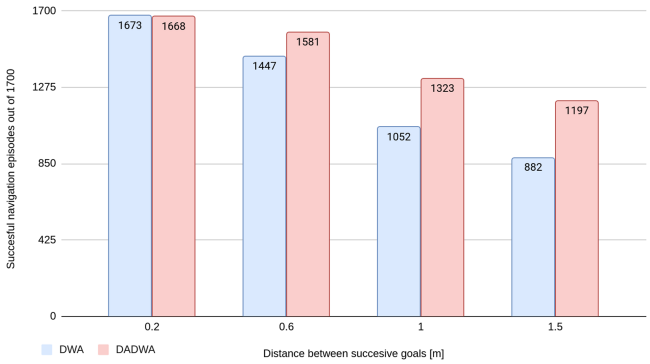

Fig. 14. Evaluation of navigation approaches, for different distances between successive goals. As the distance between goals is increased, the failures become more common. Our method shows greater robustness when compared to the DWA.

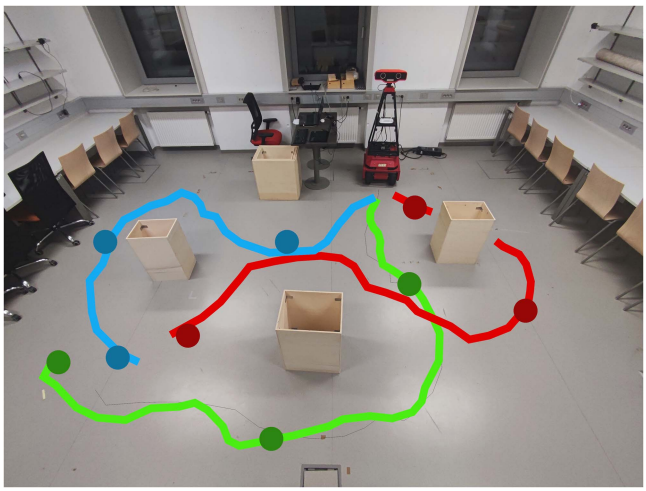

Fig. 15. Demonstration on a real robot: following sequential goals. The trained network transferred on a real robot. The lines represent the executed trajectories, and the darker circles represent the goals that were sequentially given to the robot.

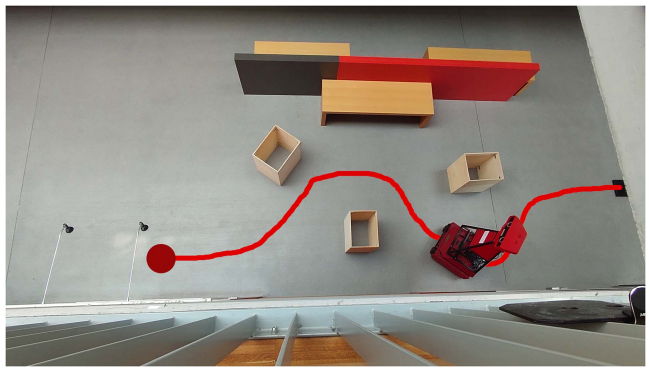

Fig. 16. Demonstration of a real robot navigating to a distant goal; the line indicates the executed trajectory.

the same visual tracking setup from the second experiment. The RGBD image of the target, the extracted goal location, and the robot coordinate frames can be seen in Fig. 17(a). The third trial was set in the identical polygonal area as the second, whereas the fourth trial took place in a modified slightly expanded polygonal area. The trajectories for these trials are showcased in Fig. 17(b) and (c). For a more dynamic representation, videos of these experiments are available in the supplementary material.

VI. Conclusion

In this article, we addressed an important problem in mobile robotics—navigation in dynamic environments. Our goal was to merge the best features of the two families of approaches that have been previously proposed to address this problem: 1) the robust, however not flexible enough, DWA local planner; 2) and highly adaptable, however less reliable, data-driven approaches. To this end, we augmented the traditional DWA cost function with an additional term that improves the performance of DWA in scenarios when moving obstacles approach the robot. Even more importantly, we proposed a neural network that predicts the parameters of the extended DWA cost function on the fly, adapting to the characteristics of the specific environment. The network is trained in an RL framework to continuously process the current and recent observations and inherently model the movement of the nearby obstacles and consequently predict the parameters that are adapted to the particular situation and its recent past. The main contribution of this article is, therefore, a new data-driven RL-based and collision-free method for local robot navigation in dynamic environments populated with moving persons.

The experimental results show that the proposed network can utilize the information in the recent observations to effectively adapt the weights of the extended DWA cost function, rendering it capable of navigating in the presence of dynamic obstacles, outperforming the DWA, as well as a deep-learning-based approach.

To further improve its performance as a local planner, the issue of stalling needs to be addressed, either by performing some recovery behavior or learning to detect those situations

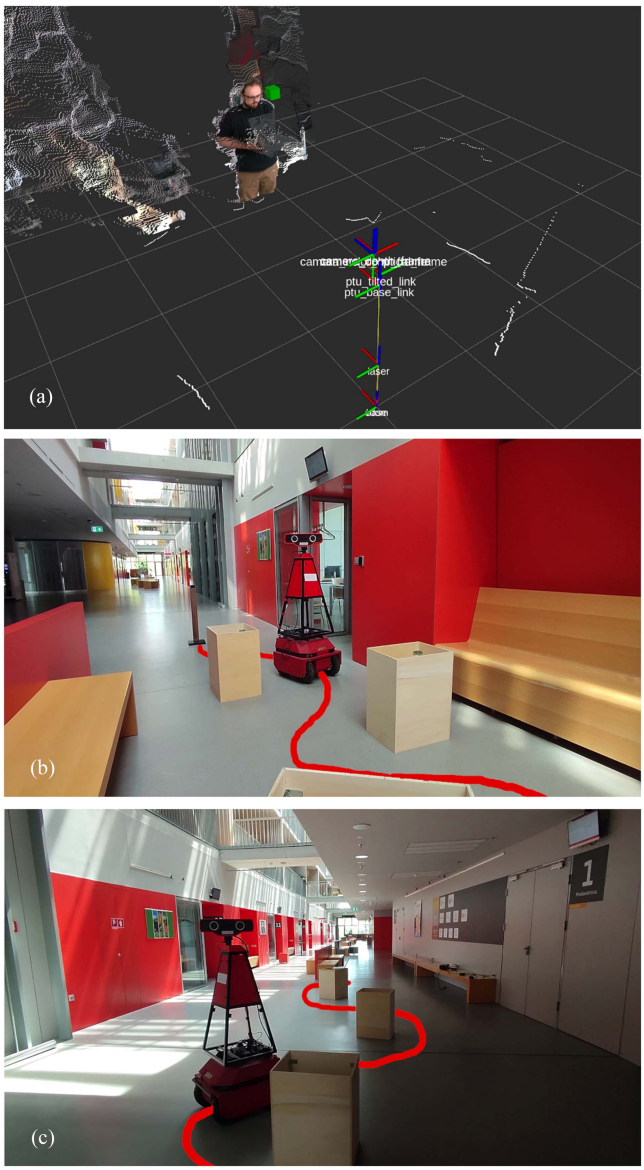

Fig. 17. Demonstration of a real robot following a moving target. (a) RGBD image of the target and the calculated goal location. (b) and (c) Executed paths in two different obstacle courses.

beforehand. Moreover, in scenarios where a map and a global planner are available, techniques that are used on top of the DWA can be implemented to further robustify the performance of DADWA and make it the best choice for a local planner in dynamic real-world navigation scenarios.

REFERENCES

[1] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag.*, vol. 4, no. 1, pp. 23–33, Mar. 1997.

[2] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 300–307.

[3] Stanford Artificial Intelligence Laboratory, "Robotic operating system," Accessed: May 22, 2024 [Online]. Available: https://www.ros.org

[4] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1999, vol. 1, pp. 341–346.

[5] M. Missura and M. Bennewitz, "Predictive collision avoidance for the dynamic window approach," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 8620–8626.

[6] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," in *Proc. RSS Workshop New Front. Deep Learn. Robot.*, 2017.

[7] Y. Zhu et al., "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3357–3364.

[8] P. Mirowski et al., "Learning to navigate in complex environments," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1059–1075. [Online]. Available: https://openreview.net/forum?id=SJMGPrcle

[9] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning navigation behaviors end-to-end with AutoRL," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 2007–2014, Apr. 2019.

[10] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 31–36.

[11] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 6015–6022.

[12] R. Guldenring, M. Görner, N. Hendrich, N. J. Jacobsen, and J. Zhang, "Learning local planners for human-aware navigation in indoor environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 6053–6060.

[13] M. Dobrevski and D. Skočaj, "Adaptive dynamic window approach for local navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 6930–6936.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[15] R. Simmons, "The curvature-velocity method for local obstacle avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1996, pp. 3375–3382.

[16] Y. N. Ko and R. G. Simmons, "The lane-curvature method for local obstacle avoidance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. Innov. Theory Pract. Appl.*, 1998, vol. 3, pp. 1615–1621.

[17] B. P. Gerkey and K. Konolige, "Planning and control in unstructured terrain," in *Proc. ICRA Workshop Path Plan. Costmaps*, 2008.

[18] P. Ogren and N. E. Leonard, "A convergent dynamic window approach to obstacle avoidance," *IEEE Trans. Robot.*, vol. 21, no. 2, pp. 188–195, Apr. 2005.

[19] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[20] M. Kobayashi and N. Motoi, "Local path planning: Dynamic window approach with virtual manipulators considering dynamic obstacles," *IEEE Access*, vol. 10, pp. 17018–17029, 2022.

[21] U. Patel, N. K. S. Kumar, A. J. Sathyamoorthy, and D. Manocha, "DWA-RL: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 6057–6063.

[22] Z. Pan et al., "D ^{2} WA "dynamic" DWA for motion planning of mobile robots in dynamic environments," *Int. J. Dyn. Control*, vol. 11, pp. 3136–3144, Dec. 2022.

[23] L. Chang, L. Shan, C. Jiang, and Y. Dai, "Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment," *Auton. Robots*, vol. 45, pp. 51–76, Jan. 2021.

[24] Z. Hong, S. Chun-Long, Z. Zi-Jun, A. Wei, Z. De-Qiang, and W. Jing-Jing, "A modified dynamic window approach to obstacle avoidance combined with fuzzy logic," in *Proc. IEEE 14th Int. Symp. Distrib. Comput. Appl. Bus. Eng. Sci.*, 2015, pp. 523–526.

[25] D. Teso-Fz-Betoño, E. Zulueta, U. Fernandez-Gamiz, A. Saenz-Aguirre, and R. Martinez, "Predictive dynamic window approach development with artificial neural fuzzy inference improvement," *Electronics*, vol. 8, no. 9, 2019, Art. no. 935.

[26] F. Leiva, K. Lobos-Tsunekawa, and J. R. Solar, "Collision avoidance for indoor service robots through multimodal deep reinforcement learning," in *Proc. 23rd RoboCup Int. Symp.*, 2019, pp. 140–153.

[27] A. Devo, G. Mezzetti, G. Costante, M. L. Fravolini, and P. Valigi, "Towards generalization in target-driven visual navigation by using deep reinforcement learning," *IEEE Trans. Robot.*, vol. 36, no. 5, pp. 1546–1561, Oct. 2020.

[28] A. Francis et al., "Long-range indoor navigation with PRM-RL," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1115–1134, Aug. 2020.
[29] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *Int. J. Robot. Res.*, vol. 35, no. 11, pp. 1289–1307, 2016, doi: 10.1177/0278364915619772.
[30] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 3052–3059.
[31] F. Xia, A. R. Zamir, Z.-Y. He, A. Sax, J. Malik, and S. Savarese, "Gibson ENV: Real-world perception for embodied agents," in *Proc. Comput. Vis. Pattern Recognit.*, 2018, pp. 9068–9079.
[32] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, no. 5, pp. 4282–4286, May 1995, doi: 10.1103/PhysRevE.51.4282.
[33] A. Lukežič, T. Vojíř, L. Č. Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter tracker with channel and spatial reliability," *Int. J. Comput. Vis.*, vol. 126, pp. 671–688, 2018.

Matej Dobrevski received the B.S. and M.S. degrees in electrical engineering from Ss. Cyril and Methodius University in Skopje, Skopje, North Macedonia, and the Ph.D. degree in computer science from the University of Ljubljana, Ljubljana, Slovenia.

He is currently a Researcher with the Visual Cognitive Systems Laboratory, Faculty of Computer and Information Science, University of Ljubljana. He works on robot navigation and computer vision projects.

His research interests include reinforcement learning in robotics, deep learning in robotics, and computer vision in general.

Danijel Skočaj (Member, IEEE) received the Ph.D. degree in computer and information science from the University of Ljubljana, Ljubljana, Slovenia, in 2003.

He is currently a Full Professor with the Faculty of Computer and Information Science, University of Ljubljana, and heads the Visual Cognitive Systems Laboratory. He has led or participated in numerous projects in his research fields, including EU projects, national research projects, and industry-funded applied projects. His work in research and development projects aims to translate research findings into practical applications. He is interested in the ethical implications of artificial intelligence, machine learning, and robotics, and their impact on society. His research interests include computer vision, deep learning, and cognitive robotics.

Dr. Skoaj has served as the President of the IEEE Slovenia Computer Society and the Slovenian Pattern Recognition Society.