State Representation Selection for Mapless Autonomous Navigation with Deep Reinforcement Learning

Alexandre G. Caldeira, Kevin B. de Carvalho, and Alexandre S. Brandão

Abstract—Mapless autonomous navigation is an open problem within Robotics, with ample academic interest and wide market growth expectations. In this task, an agent must navigate the environment safely and reliably achieving a final state, by selecting appropriate actions. Deep Reinforcement Learning methods, with focus to Double Deep Q-Networks, have shown efficient results above human-level in this task. However, most methods currently require manual state modeling, and there is no consensus to the optimal state representation given a desired performance. We propose a framework towards fine-tuning depth state representation for task-related metrics. Ensuing empirical results, conducted in simulated environments, suggest that averaging depth samples as a discrete set of states converges , with the trade-off that safety reward shaping must be rigorous. Applications and improvements are proposed and the code, data and trained models are made open source.

Index Terms—Deep Reinforcement Learning, Mapless Autonomous Navigation, Depth Sensor State Representation

1 Introduction

Autonomous navigation is an open problem within the growing demand for autonomous vehicle applications. Currently available solutions range from reactive to deliberative strategies, and are based on how much - if any - information is known *a priori* about the environment. Therefore, the main goal is to safely and reliably travel from a starting point to a final destination without human aid.

In this regard, navigation strategies face challenges in collision-free movement, environment modeling, sensor fusion, energy management, among others. Therefore, a central question intrinsically posed in autonomous navigation is: given actions and sensors available for the agent, what navigation policy reliably reaches the final destination while complying to due safety across any environment? In short, many answers have been posed to this question, using either - or a mix of - deterministic and heuristic approaches.

Mapless autonomous navigation has faced momentous advances in the recent years, fuelled by academic and industry interest. An efficient approach is the Reinforcement Learning solution to that task. Yet, it is a field under development such that different methodologies have arisen to address its shortcomings.

Evidently, most models currently depend on manual modeling for the environment, state and action, and this work focuses on the problem of agent state representation modeling. As a brief assessment and introduction to this growing field, this section presents the demands that motivated this work as well as a review of latest methods in order to set the foundations for objectively discussing our methodology, intent and contributions.

- *A. G. Caldeira, K. . de Carvalho, and A. S. Brandão are with the Núcleo de Especialização em Robótica (NERO), at the Department of Electrical Engineering of the Federal University of Viçosa (UFV), Viçosa - MG, Brazil. E-mail: alexandre.caldeira@ufv.br*

This work has been supported by CNPq, CAPES, FAPEMIG and MCTI.

1.1 Context and Motivation

Autonomous navigation technology, with its diverse applications, is a rapidly evolving field that holds significant potential for growth across various sectors. In the realm of e-commerce and warehousing, autonomous systems can enhance logistics efficiency by autonomously navigating warehouses, picking, and packing orders [1].

In the agricultural sector, autonomous navigation is being utilized for tasks such as planting and harvesting, thereby increasing efficiency as agriculture becomes increasingly mechanized [2]. Lastly, in the field of space exploration, autonomous navigation systems are being used for exploration and surveying celestial bodies [3].

1.2 Deep Reinforcement Learning

An intersectional problem in Robotics is the inherent dynamics of real world environments, which currently challenges research efforts for generalized solutions, especially in the task of autonomous navigation strategies. One efficient mathematical framework for representing the chain of decision-making related to navigation is the Markov Decision Process (MDP). Originally published by Richard Bellman in 1954 [4], MDP is a method of describing a chain of decisions as a directed graph where each node represents a state or action, and each edge describes the probability that an action will lead to a state.

Thus, MDP proposes using dynamic programming to determine a policy such that despite partly random connections between states and actions, decisions can be made to reach a desired final state. An important assumption in MDP is that the environment is a random process where the future results depend only of the current states, disregarding the past ones.

In this context, Reinforcement Learning (RL) serves as a methodology for computing the optimal policy based on expected results for each possible action in a given state. The term "Reinforcement" arises from the fact that this expected result is consolidated as an objective function known as the reward function, which inhibits or encourages decisions. Namely, by defining a finite or infinite set of states $\mathcal{S}$ and actions $\mathcal{A}$, a "Q-value" is estimated as the long term results of each action-state pair, based on a task or process-related definition of the reward function [5], [6].

As such, Q-Learning is an algorithm within RL that represents the map $Q : \mathcal{S} \rightarrow \mathcal{A}$ as a matrix to be computed with optimization methods. Then, selecting the maximum $Q(s, a)$ is a policy that ensures reaching the desired state using a critical path within the MDP process graph. This yield or overcame human-level decision making in various tasks, gaining traction in both industry and academia.

However, in close similarity to the trade-off of performance versus data sample size between Random Forests and Deep Neural Networks [7], Q-Learning suffers from tendencies to overfit, resulting in limited performance in complex or dynamic environments [8]. In this context, the Deep Q-Network (DQN) [9] was the first neural network model used to succesfully estimate $Q(s, a)$ and overcome human performance on tasks, which also resulted in a solution to the limited performance of tabular Q-Learning. Recent developments in DQN have presented strong evidence to the safety, consistency and efficiency of this approach when compared to Q-Learning [9].

DQN tends to overestimate Q-values due to its method for computing losses for gradient descent. In order to converge, the DQN θ network computes its loss as:

$$\text{Loss}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', \theta) - Q(s, a; \theta^{-})\right)^2\right]$$

r : is the reward for the current state

γ : is a discount factor to future rewards

a, a' : are the initial and next actions

s, s' : are the initial and next states

where θ is the network for estimating Q, while θ^{-} is a copy of θ made before the current training. This difference between the *a priori* ($Q(s, a; \theta^{-})$) and *a posteriori* ($Q(s', a', \theta)$) reward estimation is known as Temporal Difference (TD) error.

The DQN method for computing the TD error results in systematic overestimation of rewards, and an approach to solve this issue was provided by Hado van Hasselt [10], proposing that two networks are used in the learning process, entitled the Double Deep Q-Network (DDQN) method.

In DDQN, one network is trained and used to estimate Q-values in order to select actions, and will be regarded in this paper as the "Selector θ" network. Another network, named here as the "Evaluator θ^{-}", is used for estimating more accurate Q-values *a posteriori*, such that the DDQN loss equation is given by:

$$\text{Loss}(\theta) = \mathbb{E}\left[\left(r + \gamma Q(s', a'(\theta); \theta^{-}) - Q(s, a; \theta)\right)^2\right]$$

$$a'(\theta) = \arg\max_{a'} Q(s', a'; \theta)$$

such that the Selector θ chooses both the current and next actions (a, $a'(\theta)$) but the Q-value of next action is computed by Evaluator θ^{-}. This Q-value evaluated by θ^{-} is then subtracted from the Q-value originally estimated by Selector θ, computing a more stable TD error, thus also a more stable loss and convergence. Similarly to DQN, in DDQN the Evaluator θ^{-} is never trained, only copied from Selector θ after a chosen amount of training episodes.

In practical terms, this means that errors by Selector θ are computed comparing its *a priori* Q-value with the *a posteriori* Q-value estimated by an older copy using the originally chosen actions, reducing overestimation. DDQN has shown better results than DQN in terms of training length in time as well as resulting performance, and was therefore selected.

1.3 State Representation in DRL

Despite the compatibility of most DRL algorithms with discrete sets of states and actions, there have been efforts to extend this to continuous state and action spaces. This is particularly important for depth-based sensors such as Light Detection and Ranging (LiDAR) instruments and RGB-D cameras, which provide continuous state spaces. Larger state spaces incur in more possible Q-values, leading to longer training in DRL, however there is no consensus to the minimum optimal depth state representation [11], [12].

In terms of similar works, different applications have used DDQN, and even compared DDQN and DQN performance in the task of mapless autonomous navigation [8], [13], [14]. However, these do not address the problem of state representation adjustment, defining a larger state space then our work, despite the authors addressing them as low-dimensional, as well as different agents and actions [15], [16], [17]. Previous works in similar applications have used DRL for ground and aerial unmmaned vehicle navigation successfully, as well as in the of context path planning and autonomous object handling [18], [19]. These also do not regard the problem of state representation, focusing on the task fulfillment rather then discussing state modeling objectively.

1.4 Goals, Contributions and Impact

Currently, to the best of the authors' knowledge, there are no other works objectively discussing the impacts of depth state representation in the performance of DRL agents. Closing this gap with strong evidence enables better informed modeling for future works in the regard of minimum sensing required for convergence as well as expected performance in both training and testing. Our contribution is threefold, and lies in (i) a state representation method is selected based on the best performing DRL model for mapless autonomous navigation, therefore laying (ii) a framework for perfomance-oriented state representation comparison and (iii) training convergence and test performance trade-offs between different representation methods. This enables and fuels further work towards analytically deriving the minimum optimal state representation for a given task in DRL. All datasets and models used or produced in this work are made Findable, Accessible, Interoperable and Reusable (FAIR), providing also open source code for future replication, following best practices [20], [21].

2 The Simulation Setup

This section describes a replicable and flexible communication, agent, sensing and environment definitions used for training and testing our models for analyses. The infrastructure for testing the autonomous agents was established through the selection of open source tools, guided by successful related work and previous results from our lab [18].

2.1 Communication framework

Robot Operating System (ROS) is an open source framework that is widely used in both academic and industry context for the development and deployment of autonomous agents. At its core, ROS functions as an automated and structured message broker middleware over TCP connections. This allows publishing and listening to different nodes that may represent sensors, robots, back-end services, automated models or any other piece of hardware with internet connection. Nodes have topics, messages that represent their capabilities such as the odometry sensors of a robot, or velocity commands that can be published to an agent, and so forth for different contexts.

This structure grants ROS great flexibility and interoperability, allowing seamless transition of models, agents and sensors: all the code needs to assure is that the correct topic is being read or published over the network, regardless of what hardware receives or sends the topic. Moreover, this means that changing the selected agent or sensor, even migrating from simulations to a real environment is made possible by ROS with hardware-agnostic communication and code development. The code for the setup described in this section, and further sections, was made publicly available in the following GitHub repository: https://github.com/Alexandre-Caldeira/sia_DRL_2023 .

2.2 Environment, agent and sensor

Applying ROS as a communication framework on the open source Gazebo physics simulator enables automating computationally expensive simulations, which also required building different environments containing obstacles for avoidance by the agent. To implement this, we select a widespread-use ground robot and 2D-LiDAR, similar to ubiquitous equipment from academic publications and industry applications in the field of Robotics.

2.2.1 Simulation Environments

Three simulation environments were designed for mapless navigation simulations to train a model and measure its performance in new territory. Figure 1 illustrates these environments, containing obstacles of varying shapes, in different positions, orientations and distances from each other. Environment 1 is shown in Figure 1, at the top-left corner, featuring one base point location where the agent starts each simulation (B1), as well as two goal locations (G1) and (G2). The goal (G2) is designed for verifying that the agent did not overfit and memorize the path to (G1), since only (G1) will be used in training.

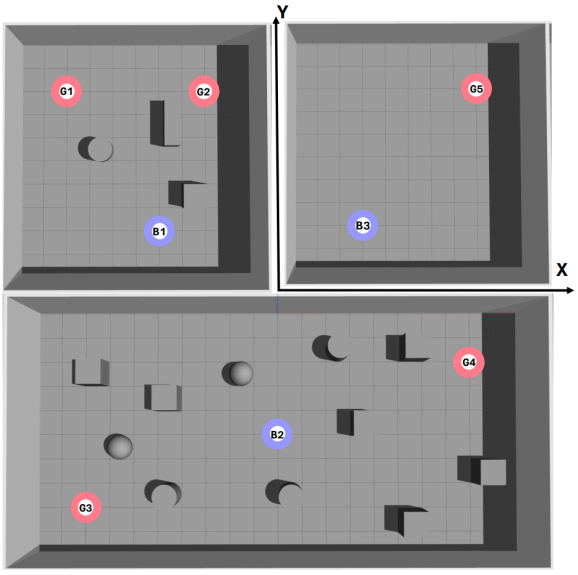

Fig. 1. Simulation environments for training the agent and testing its performance, where starting bases are shown in blue and ending goals are indicated in red.

In the bottom of Figure 1, Environment 2 similarly shows a starting point (B2) and two different goal points (G3) and (G4), that will be used to determine whether the agent is able to navigate in a larger, previously unseen map after being trained. Finally, at the top-right corner of Figure 1 displays a final base point (B3) and goal point (G5), in a room identical in size to Environment 1, however containing no obstacles. The purpose of Environment 3 is to validate that the agent is able to navigate environment without reference points from obstacle shapes.

Pioneer 3-DX is a ground robot designed by Adept Mobile Robots, capable of achieving 1.2 m/s in linear velocity and 300°/s angular velocity. It has two front wheels with differential drive motors, and a third stabilization non-controllable back wheel. The model for this robot is available in Gazebo and a ROS-based communication handler class was developed for measuring the position and orientation of the agent in the environment, as well as for sending linear and angular velocity commands.

As for the sensing unit, a 2D-LiDAR depth measurement instrument available publicly for use in Gazebo was chosen. Its measurements are returned within ROS communication with a 270° field of view, which is reduced in this work to 727 measurements in a 0 to 5-meter range, from 0 to 180°, centered at the front of the agent.

3 Methodology

Given the definition of agent, environment and sensor, the next step for developing an autonomous navigation strategy is defining a model to encompass safe and reliable solutions to the task. This section presents our proposal for collecting and comparing results of training and testing agents under Double Deep Q-Network (DDQN) modeling with different Depth State Representation Methods (DSRMs).

Environment representation using depth sensors is a topic that currently lacks academic consensus in both real and simulated settings. This uncertainty extends to deterministic and heuristic models, as noted in the literature [22]. In this study, we aim to pave the way for performance-driven representation modeling decisions, focusing on autonomous navigation with DDQN. Our methodological approach is designed for repeatability across various methods, thus contributing to a more extensive body of evidence, which we hope will aid in the eventual determination of optimal environment representation based on sensor data for specific desired metrics.

In the training environment, as shown in Figure 2, the initial position and orientation are defined for the agent training, seen also as (B1) in Figure 1. Here, a depth sensor measures the distance between the agent and objects within the environment, which must be modeled to - ideally - ensure autonomous navigation with safety guarantees. Figure 3 displays the complete readings of the depth sensor in this state, covering a range of 180 degrees centered in front of the agent and collecting 727 measurements within a 0 to 5-meter depth range.

Furthermore, Figure 3. separates the readings into 4 sectors, each spanning 46 degrees on the topmost image. Below, Figure 3 similarly presents the same readings divided into 10 intervals of 18 degrees each. A comparative analysis of the content within each sector in Figure 3 reveals that different details captured within the same initial measurement. Specifically, Figure 4 showcases sector #2 from the topmost image in Figure 3 and the 4th sector on the lower image, capturing the edge of the wall through the collected measurements and the histogram of this sector.

Definition 3.1. Depth State Representation Methods: DSRMs are quantizations of the sensor sample distribution of a number of sectors by a statistical measure. In our work we selected the mean, mode, and the minimum value of the twentieth percentile, henceforth referred to as ”soft_min”, as DSRMs to be used for 4, 5, 6 and 10 sectors.

Notably, when comparing the readings in Figure 4 it becomes evident that the same edge, when quantified through different sampling, results in distinct measurements of the same environmental phenomenon. Consequently, this study proposes defining the agent’s state based on the separation of LiDAR measurements into 4, 5, 6, and 10 sectors, using the mean, mode, and ”soft_min”. Ensuing sections will then describe how, by varying the DSRM, the number of states and their rendition of the environment result in differently shaped DDQN models to be trained and compared in tests.

3.1 Double Deep Q-Network Model

In order to model the mapless autonomous navigation task as a MDP for solutions with DDQN, states and actions must be clearly defined for the agent, in order to set parameters for the network architecture and the rewards that will inhibit or encourage specific behaviors related to the task. This section defines discrete sets of possible states and actions, a neural network architecture compatible with these sets, and finally defines rewards, based on the task.

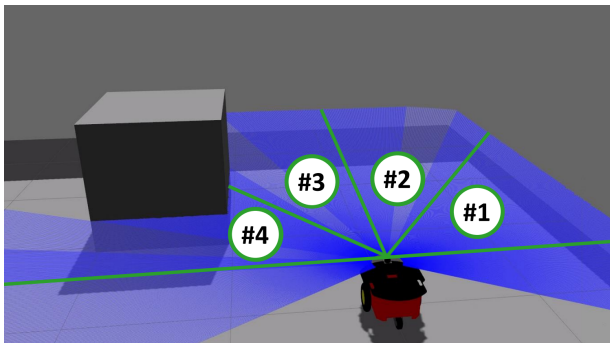

Fig. 2. Depth scans are selected from -90° to 90°, centered in front of the robot, and split into 4 sectors with equal sample size.

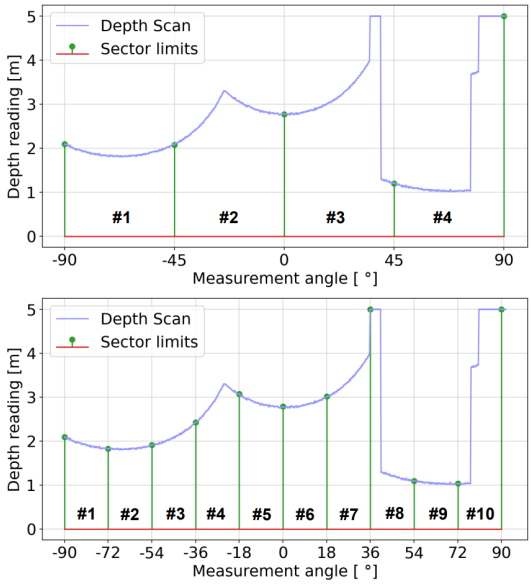

Fig. 3. LiDAR depths scans are split into 4 and 10 sectors respectively, resulting in sample sizes in each sector for the same measurement.

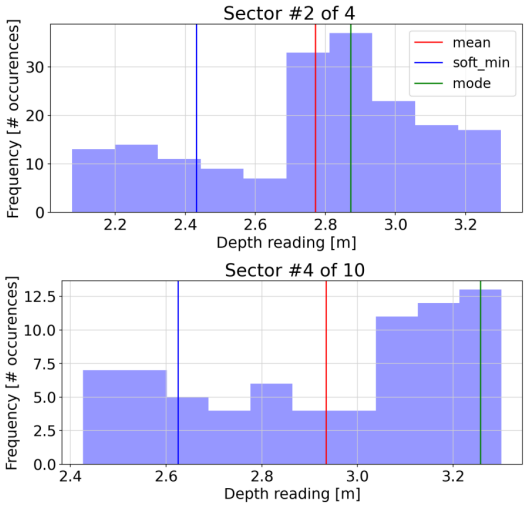

Fig. 4. Selecting sector #2/4 and #4/10 demonstrates that the same edge on the wall (seen in Fig. 2) results in varying distributions and therefore different mean, mode and soft_min values.

3.1.1 State and action spaces

Building upon the DSRM definitions, the states that the agent uses to perceive the environment are defined as a discrete set of numeric values. Moreover, in order to encourage the agent to be oriented towards the goal, an extra goal-orientation directional state $\mathcal{S}_o$ is defined by splitting the depth agent orientation into 6 states. Namely, an integer is used to represent whether the goal is behind, or in one of five directions in front of the agent.

Definition 3.2. $\mathcal{S}_o$: the goal-orientation directional state, is designed as:

$$\begin{aligned}\mathcal{S}_o = \Big\{ & S_{back}, S_{[-90,-53°]}, S_{[-54,-17°]}, \\ & S_{[-18,17°]}, S_{[18,53°]}, S_{[54,90°]} \Big\} \\ = & \{0,1,2,3,4,5\} \in \mathbb{N}^1\end{aligned}$$

such that, $\mathcal{S}_o = 0$ if the goal is behind the agent, $\mathcal{S}_o = 1$ if the goal is at the LiDAR angle interval between $[-90, -53°]$, and so on for the other intervals.

Definition 3.3. $\mathcal{S}_d$: the set of depth states $\mathcal{S}_d$ is modeled as a discrete set of sectors with 4, 5, 6 or 10 elements, each of which are real numbers computed with a given statistical measure, composing a DSRM: $\mathcal{S}_d \in \mathbb{R}^{\{4,5,6,10\}}$.

Finally, the complete state space for the agent is defined as the union of the set of DSRM samples and goal-orientation directional state: $\mathcal{S} = \mathcal{S}_d \cup \mathcal{S}_o$.

$$\text{Agent Pose: } \begin{cases} \xi = [x, y] \\ \psi = \arctan(y/x) \end{cases} \tag{1}$$

Therefore, three actions are selected for the agent to navigate the environments. Namely, the agent can either move straight forward with linear velocity, or make a soft curve to its left or right, composed by a linear and angular velocity. Thus, we define the action space as a discrete set of three actions, mathematically given by:

$$\mathcal{A} = \left\{ \begin{array}{ll} a_{right} & = (\dot{\xi} = 0.2 \text{ m/s}, \dot{\psi} = -0.4 \text{ rad/s}), \\ a_{forward} & = (\dot{\xi} = 0.4 \text{ m/s}, \dot{\psi} = \quad 0 \text{ rad/s}), \\ a_{left} & = (\dot{\xi} = 0.2 \text{ m/s}, \dot{\psi} = +0.4 \text{ rad/s}) \end{array} \right\} \tag{2}$$

Thus, the agent action and state spaces are formally defined, where the action set is $\mathcal{A} \subset \mathbb{R}^{3\times 2}$ and the state set is $\mathcal{S} \subset \{\mathbb{R}^{\{4,5,6,10\}+1} \cup \mathbb{N}^1\}$.

3.1.2 DDQN Architecture

In terms of neural network architecture, a shallow multi-layered perceptron has been shown to yield sufficient performance, and was thus selected as minimal architecture in this work. Namely, it is defined as the sequential application of activation functions of inputs multiplied by weights and biases matrices, representing the oriented graph for the network, shown in Figure 5.

In this architecture, the input state s is processed through three fully connected layers with the 'Leaky' ReLU activation function. The first layer (θ_1) maps the input to a 64-dimensional hidden representation, the second layer (θ_2) further transforms this representation in a 64-dimensional space, and the third layer (θ_3) maps it to a 3-dimensional output representing the Q-values for each of the 3 available actions. Mathematically, outputs from the network seen in Figure 5 are given by:

$$Q(s,a) = f_{\text{L-ReLU}}\left(\theta_3 \cdot f_{\text{L-ReLU}}\left(\theta_2 \cdot f_{\text{L-ReLU}}\left(\theta_1 \cdot s\right)\right)\right) \tag{3}$$

where:

$$\begin{aligned} s &: \text{Input state vector of size } \{4,5,6,10\}+1 \\ a &: \text{Action vector of size } 3 \\ \theta_1 &: \text{Input weights, shaped } (\{4,5,6,10\}+1) \times 64) \\ \theta_2 &: \text{Hidden layer weight matrix, shaped } (64 \times 64) \\ \theta_3 &: \text{Output layer weight matrix, shaped } (64 \times 3) \\ f_{\text{L-ReLU}} &: \text{'Leaky' Rectified Linear Unit activation} \end{aligned}$$

This architecture learns to estimate Q-values for state-action pairs, therefore computing the expected sum of future rewards for selecting actions in mapless navigation.

3.1.3 Reward function shaping

The reward function was modeled based on reliability and efficiency requirements for the task of autonomous navigation. In detail, five rewards are presented below, composing the total reward for any given new state after acting.

Definition 3.4. R_{dist}: distance reward, is a reward was given based on how closer the agent is to the goal at each iteration of an episode. The reward fraction R_{dist} is calculated as:

$$R_{dist}(i) = k_{R_{dist}} \cdot \frac{|D_{start} - D_i|}{D_{start}} \tag{4}$$

where the initial and current distance to the goal are given by:

$$\begin{aligned} D_{start} &= \|\xi_{start} - \xi_{goal}\| \\ D_i &= \|\xi_i - \xi_{goal}\| \end{aligned}$$

such that the reward for staying at the starting position is null and increases as the agent reaches the destination in every iteration step i within an episode. The constant

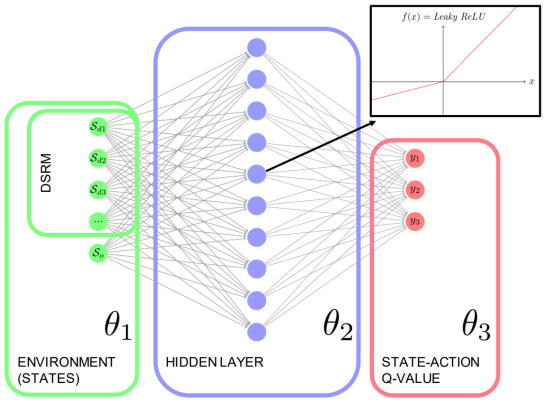

Fig. 5. A fully connected network is used to estimate Q-values for each of the 3 possible actions, given a state.

$k_{R_{dist}}$ is a free parameter that was set to 80 analytically, to yield null rewards after a certain number of iterations, related to the negative reward R_{steps}. Note that, because of the modulus function, this reward does not inhibit straying further from the goal, which is addressed later in R_{steps}.

Definition 3.5. R_{col}: collision reward, is a negative reward built for inhibiting the agent from navigating close to objects, other environment components or crashing. For each sector, the depth sample distance in each iteration ($d_{s,i}$) is discretized as close (≤ 0.5 m), acceptable ($\leq 1,3$ m) or safe, and negative rewards are attributed to each case:

$$R_{col}(i) = \sum_{s=1}^{N_{sectors}} \left\{ \begin{array}{ll} -100, \text{ if} & d_{s,i} \leq 0.5 \text{ m} \\ -2, \text{ if} & 0.5 < d_{s,i} \leq 1.3 \text{ m} \\ -1, \text{ if} & d_{s,i} > 1.3 \text{ m} \end{array} \right\} \quad (5)$$

Definition 3.6. $R_{success}$: success reward, is a was given for reaching the vicinity of the goal. Numerically:

$$R_{success}(i) = \left\{ \begin{array}{ll} k_{R_{success}} & \text{, if } D_i \leq d_{reach} \\ 0 & \text{, if } D_i > d_{reach} \end{array} \right. \quad (6)$$

where $k_{success}$ is a free constant parameter, set to 20. The minimal distance from the agent to the goal - used to consider the episode a success - is defined as d_{reach}. It is also a free constant parameter that was set to 0.5 meters in our simulations.

Definition 3.7. R_{steps}: steps reward, is a negative reward received for each iteration, thus encouraging the agent to minimize the number of steps taken to finish its navigation. This reward was computed as:

$$R_{steps}(i) = -k_{steps} \cdot i \quad (7)$$

were k_{steps} is a constant positive free parameter, set analytically to 2, such that a reward identical to a collision would be given for every step after iteration number 50.

Definition 3.8. R_{dir}: goal direction reward, is a reward was proposed for encouraging the agent to align its direction to the goal by using the goal orientation state. If the orientation state $\mathcal{S}_o$ is aligned to the goal a positive reward is given, and a negative reward is received otherwise:

$$R_{dir}(i) = \left\{ \begin{array}{ll} -k_{dir,1} & \text{, if } \mathcal{S}_o \neq 3 \\ k_{dir,2} & \text{, if } \mathcal{S}_o = 3 \end{array} \right. \quad (8)$$

where $\mathcal{S}_o$ is the goal orientation state present in all methods, $k_{R_{dir}1}$ is a positive free parameter set to 3, and $k_{R_{dir}2}$ is a positive free parameter set to 10 to match other reward amounts.

Finally, the total reward for each iteration of episode is calculated as the sum of the five previous reward. Therefore, a total reward:

$$R_T(i) = R_{dist} + R_{col} + R_{success} + R_{steps} + R_{dir}$$

was given after reaching a new state, at each iteration sequence of action selection after a state observation.

3.2 Algorithm for Training and Testing

Following the definition of the training and testing environments, using the agent states and possible actions, an algorithm was defined for training the DDQN model. In detail, the learning phase is split into training and validation where the convergence of the model for each given DSRM is analysed.

In DRL, mapless autonomous navigation is achieved by means of action selection based maximizing expected rewards. That is, in practical terms, selecting the action that has the highest Q-value for a state-action pair and therefore the highest expected rewards.

In Figure 6, the complete loop for autonomous navigation is shown, where the agent perceives the environment through its DSRM, generating a state s. Inputting this state into the Selector network computes Q(s,a) for each action as outputs, and selecting the output with the highest Q-value results in choosing one of the three possible actions. At each decision, the agent has an $\epsilon \in [80, 0.15]$ probability of choosing a random action instead of the network output, a method known as ϵ-greedy for balancing exploration and exploitation.

This incurs in interaction with the environment by moving, which leads the agent to its next state s'. This set of state, action, new state and the Q-values (s,a,s',Q) is then saved to a memory buffer at every iteration, storing up to 50k iterations in our particular algorithm.

Definition 3.9. Episode: environment navigation sequence of at most 60 iterations of action selection based on Q-value estimation from the current state. Episodes may be shorter than 60 iterations - also named steps - in case of success or collision.

After each 10-episode interval, the simulation is paused and the agent enters the Training loop, that is perscribed within the dashed area in Figure 6. In this stage, each recording from the memory buffer has its s' (next state) inputted into the Evaluator θ^-, thus estimating future rewards from its outputs Q^-.

Then, these estimated future rewards from the Evaluator θ^- are subtracted from the rewards estimated originally by the Selector θ, resulting in a squared error that is used as

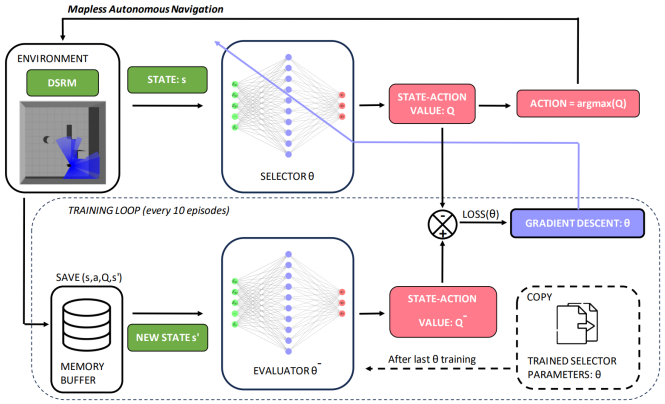

Fig. 6. A Selector network is trained for the mapless autonomous navigation task, supported by a less-often updated Evaluator network (DDQN).

loss function for applying gradient descent to the Selector θ. Effectively, this loss represents the TD error, and is used for correcting the reward expectations of the Selector θ. Next, the parameters for the Evaluator θ^- are updated by copying the newly-trained Selector θ as shown on the right side of Figure 6. Finally, the simulation resumes to navigation, using only the Selector θ for decisions, and applying the Evaluator θ^- only for the training loop.

This process of autonomous navigation lasts for 25k episodes - iteratively training at each 10 episodes - at which point the training process is interrupted regardless of its performance. Note that this training process is used for 25k episodes for each specific DSRM, resulting in 12 different models: one for each pair of number of sectors (4, 5, 6, 10) and sampling method (mean, mode, "soft_min"). The code for the algorithm was made available publicly in a GitHub repository: https://github.com/Alexandre-Caldeira/sia_DRL_2023 .

3.2.1 Training hyperparameters

During the training phase, the application of gradient descent for the learning convergence of the Selector θ requires some optimization strategy. In this work, the adaptive moment estimation "Adam" optimizer was selected [23], with a Exponential Step Learning Rate, both available publicly for use in Pytorch, in accordance with the original publication and compatible with the DDQN Arquitecture implementation from Section 3.1.2.

Mathematically, the StepLR function is a piecewise exponential decay method for the learning rate hyperparameter in neural network training. Its update rule in the present work is defined as $\alpha_{start} = 10^{-3}, \alpha_{end} = 10^{-6}, \mathrm{N_{updates}} = \mathrm{N_{episodes}} \times 0.7$, with

$$\alpha_{decay} = \left(\frac{\alpha_{end}}{\alpha_{start}}\right)^{\frac{1}{\mathrm{N_{updates}}}} \tag{9}$$

where the total number of episode $\mathrm{N_{episodes}}$ is 25k, and the definition of initial and final learning rates (α_{start}, α_{end}) were chosen empirically.

Similarly, the ϵ parameter related to random exploration of state-action pairs decays exponentially across training, and is defined as:

$$\epsilon(\mathrm{Episode}) = 80 \cdot 0.99975^{(\mathrm{Episode})} \tag{10}$$

such that the agent initially has an 80% probability to choose random actions, decaying up to

$$\epsilon_{\mathrm{final}} = 80 \cdot 0.99975^{(25k)} = 15.43\%$$

as training progresses. This training process helps adapt the learning rate during training to improve the convergence and stability of the model, as well as ensures that exploration and exploitation takes place according to ϵ-greedy.

3.2.2 Validation strategy

During training, it is common and standard practice to validate learning with neural networks by applying a new set of data that has not been previously seen and thus mirrors the current ability of the network to answer the given task. In this work, 10 episodes were used as validation at every 100 episodes of learning.

In other words, after every 100 episodes of data collection and training, the Selector network was put to validation for 10 episodes, where its performance was saved. Based on the success and collision rate during validation, in tandem with the rewards received, it is thus possible to validate whether or not the model has converged. Note that data from validation is never used for training but serves for convergence analysis.

3.2.3 Testing strategy

After training is complete, each model (for each DSRM) is put to test in order to compare performance based on the previously defined metrics. During this phase, no training is allowed, and only the Selector θ is used for navigation.

Sequentially, 1k episodes in each one of 5 tests were performed for each of the 12 DSRMs, targeted at reaching the five goals presented in the simulation environments (Figure 1). Specifically, the tests were conducted with the following pairs of bases and goals:

$$\begin{aligned}
&\text{Test 1 - Environment 1} : (B1) \rightarrow (G1)\\
&\text{Test 2 - Environment 1} : (B1) \rightarrow (G2)\\
&\text{Test 3 - Environment 2} : (B2) \rightarrow (G3)\\
&\text{Test 4 - Environment 2} : (B2) \rightarrow (G4)\\
&\text{Test 5 - Environment 3} : (B3) \rightarrow (G5)
\end{aligned}$$

Test 1 mirrors the validation steps in training, whereas Test 2 verifies that no overfit occurred, as they are set at the training Environment 1. Test 3 and 4 check the resulting generalization achieved by the model, since they are set in an unseen and larger Environment. Finally, Test 5 regards the ability of the model to minimize steps towards reaching the goal, as Environment 3 is similarly shaped to Environment 1, but contains no objects.

3.3 Performance Metrics for Analyses

Based on similar literature, a selection of metrics was used for defining the success and collision rate, as well as measures of safety and energy efficiency during navigation.

Definition 3.10. TDT: Total Distance Traveled, is a measure of energy efficiency based on path length. The sum of the distance travelled in each iteration results in the TDT over an episode, and is defined as:

$$\mathrm{TDT(Episode)} = \sum_{i=2}^{N_{iter}} \|\xi_i - \xi_{i-1}\| \tag{11}$$

where ξ_i and ξ_{i-1} are respectively the current and last positions of the agent at each iteration step.

Definition 3.11. SR: Success Rate, is the average of the amount of episodes where the agent reaches the vicinity of the goal. The Final Distance To Goal, a support measure, is defined as:

$$\mathrm{FDTG}(\xi_F, \xi_G) = \|\xi_F - \xi_G\| \tag{12}$$

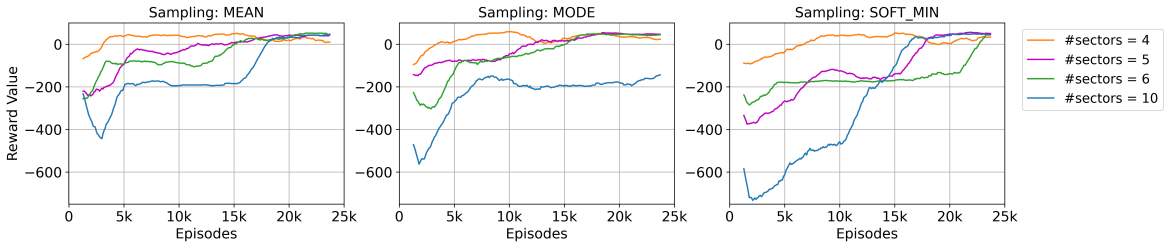

Fig. 7. Rewards results are shown for each DSRM in terms of sampling and sector number, measured during validation loops within the training stage. Initial rewards are mostly negative, arising from the collision and step reward functions, and converge to positive values as the model learns to succeed in collision-free mapless navigation. Note that mode10 DSRM fails to converge, reaching a learning plateau at -200 reward points.

where ξ_F and ξ_G are respectively the agent final position and the position of the current goal. Based on the agent's FDTG, the Success Rate is counted and averaged over episodes of each DSRM as:

$$\text{SR(DSRM)} = \frac{1}{N_{ep}} \sum_{i=2}^{N_{ep}} \left\{ \begin{array}{ll} 1 & \text{, if FDTG} \leq d_{reach} \\ 0 & \text{, if FDTG} > d_{reach} \end{array} \right\} \quad (13)$$

where d_{reach} is the minimal distance to the goal that should be considered a Success, used as 0.5 m in this work.

Definition 3.12. MDTO: Mean Distance To Obstacles, is a safety measure, defined averaging the minimal distance that the agent read to objects within the environment, and computed as:

$$\text{MDTO(Episode)} = \frac{1}{N_{iter}} \sum_{i=1}^{N_{iter}} \min_{d \in \mathcal{S}_d}(d) \quad (14)$$

where d is the distance measured and quantized in each state in $\mathcal{S}_d$, whose size depends on the DSRM in use.

Definition 3.13. CR: Collision Rate, similarly to the SR, is the average of the amount of episodes where the agent collides into objects, mathematically defined as:

$$\text{CR(DSRM)} = \frac{1}{N_{ep}} \sum_{i=1}^{N_{ep}} \left\{ \begin{array}{l} 1, \text{if } \min_{d \in \mathcal{S}_d}(d) \leq d_{safe} \\ 0, \text{if } \min_{d \in \mathcal{S}_d}(d) > d_{safe} \end{array} \right\} \quad (15)$$

where d_{safe} is the minimal distance 0.3 m in the algorithms within our work.

4 Results and Discussion

The algorithm for training and testing the 12 DSRMs over 25k episodes was applied sequentially, and the present section discusses the convergence analysis during validation loops. Next, a comparison of the performance of each model is presented based on standard metrics, followed by recommendation remarks for developing, improving and applying the models from this work.

4.1 Training convergence and validation

During training, as described in Section 3.2.2, performance data was measured for 10 episodes after each 100 episodes of training. For each of the 12 DSRMs, the reward value for each of theses episodes is presented in Figure 7. Each plot presents the results for a given sampling method, comparing the results for a increasing number of sectors. Successful model training is seen in Figure 7 as progressive growth towards positive values. In contrast, a model that fails to learn the task displays stagnant values over episodes, as seen for the DSRM mode with 10 sectors.

For the every sampling method, Figure 7 shows that DSRMs with 4 sectors reach positive rewards faster, before 8k episodes. In addition, DSRMs with 5 and 6 sectors converged in after 15k to 20k episodes in most cases and successfully converged. Moreover, DSRMs with 10 sectors begin training with the most negative amount of rewards, and required more episodes to converge than other methods. Furthermore, the DSRM "mode_10" failed to converge, as seen on the blue line of the top-right plot in Figure 7,

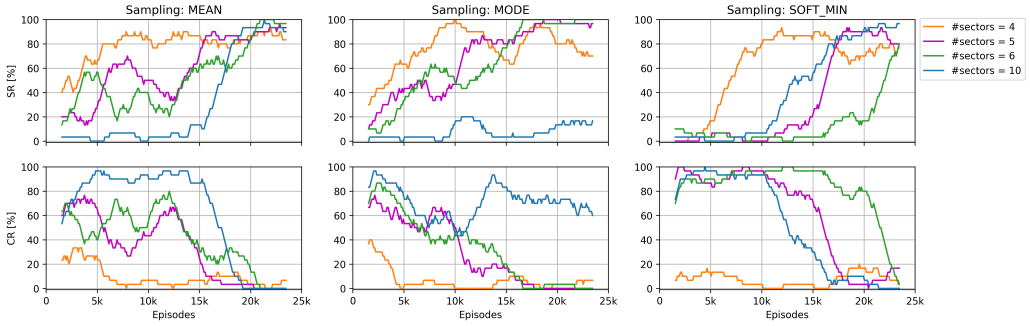

Fig. 8. Success and collision rates are seen for each DSRM over 25k training episodes. Model training is shown to result in less collisions and more successes as desired. As in Fig. 7, mode10 failed to converge as noted as low SR and high CR values.

reaching a plateau after 10k episodes. It is possible that adjusting hyperparameters such as the DDQN architecture, ϵ decay, learning rate decay, maximum number of episodes and the episode interval between training loops may result in convergence of the "mode_10" DSRM model.

Overall, increasing the number of sectors resulted in a increased number of episodes for achieving stable positive results, representing a stage where navigation is successful. In terms of success and collision rates, similar behaviour to the reward convergence is observed, where the colision rate tends to 0% as the success rate tends to 100% (Figure 8) at the same episode range where rewards converged to positive values. In short, most DSRM converged and reached similar final rewards, SR and CR at later episodes (20k to 25k).

TABLE 1
Median of test metrics averaged over 1k episodes for each DSRM demonstrate that mode sampling displays the most consistent adjustment, such that mode10 is the ideal DSRM with similar results to the top performing model (*softmin5*).

DSRM	Success Rate [%]	Collision Rate [%]	Total Distance Traveled [m]	Mean Distance To Objects [m]
mean4	29	67	19.66	1.29
mean5	65	34	21.23	1.31
mean6	70	28	20.41	1.30
***mean10**	**90**	**10**	**21.18**	**1.34**
softmin4	30	60	27.47	1.21
softmin5	***92***	***5***	***22.80***	***1.43***
softmin6	69	22	20.07	1.20
softmin10	81	15	20.89	1.37
mode4	27	73	21.23	1.24
mode5	40	60	20.17	1.32
mode6	10	87	21.59	1.23
mode10	0	98	15.14	0.97

4.2 Performance testing

Upon completion of the training process, each of the 12 models were applied to five tests as presented in Section 3.2.3, resulting in a total of 1k episodes of navigation in a mix of seen and unseen base points, goal points and environments. The mean value across all five tests was computed in a centered rolling window of 25 episodes. Figure 9 presents the performance achieved by each model, where the mode of the result is shown in red and the standard deviation is colored based on the number of sectors of each DSRM, for ease of comparison. In addition, Table 1 presents a numerical summary of the data show in Figure 9. Therefore, in short, DSRMs using mean sampling display consistent performance progression such that mode10 is the best performing model, despite similarities to the softmin5 DSRM in Success and Collision Rate.

In terms of best performance, the "soft_min" DSMR with 5 sectors displayed the smallest standard deviation across SR and CR, achieving 92% mean success rate over all tests, despite not presenting the best TDT as seen in Figure 9. Furthermore, this method achieved the highest mean MDTO, meaning that the agent is able to keep a larger distance to obstacles in general while successfully fulfilling the task.

The second-to-best performance was achieved by the "mean" DSMR with 10 sectors, resulting in 90% mean success rate, despite larger standard deviation when compared with the "soft_min5" DSRM. Regarding the "mean" sampling method over its variations, notice in Figure 9 that the SR and CR increases monotonically as more sectors are added, which is not observed for other measures. In addition, the "mean" sampling method achieved similar mean TDT regardless of the number of sectors, but an increasing mean MDTO as the number of sectors increases.

In relation to the "mode" sampling method, mirroring training results, the metrics show that this method presents the least stable performance, displaying high standard deviation across all measures. Also, note that the "mode6" DSRM failed to navigate new environments (SR = 1%, CR = 96%), despite having shown convergent behaviour in training. Similarly, the "mode10" DSRM resulted in the worst mean performance across all metrics, having failed to converge during training.

Overall, the mode sampling method shows overfit behaviour, having low success and high collision rates. The "soft_min" sampling method achieved the top performance across all DSRM when using 5 sectors, but has high variability when adding or removing sectors, making it a unstable and counter-intuitive sampling method.

Moreover, the "mean" sampling method showed very similar results in its best performing DSRM, and sector addition monotonically improves performance across mean

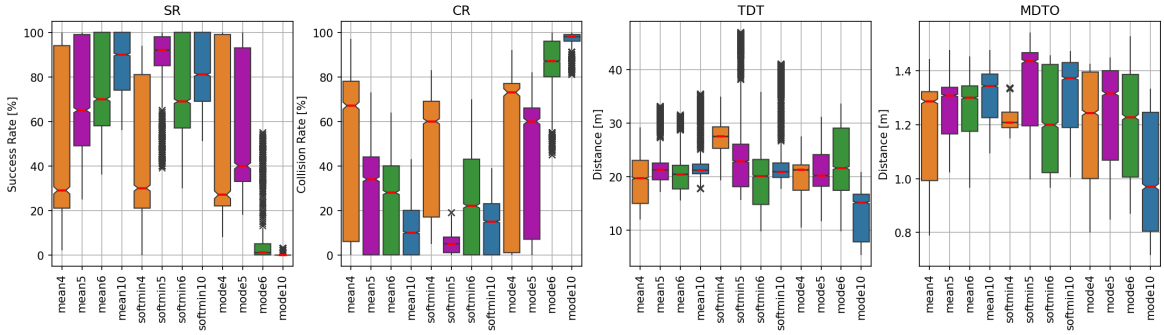

Fig. 9. Model performance in 5 tests is averaged over 1k episodes and measured on Success Rate (SR), Collision Rate (CR), Total Distance Traveled (TDT) and Mean Distance to Objects (MDTO) from left to right. Increasing sector number resulted in SR and decreasing CR for the mean sampling methods, while softmin and mode had varying results. The top individual performing model is softmin5, resulting in the highest SR, lowest CR and highest MDTO, representing respectively a successful and safe model. However, mean sampling is considered the best representation methodology following from its performance increase consistency based on sector number adjustment.

metric results. Note, also, that the "mean10" results yield similar performance to the "soft_min5" in all metrics, and as expected, "mean10" displays lower TDT at the cost of a higher MDTO.

In practice, this means that the "mean" sampling method is a more stable DSRM to tune and has the overall best convergence adjustment. Still, it requires more attention to safety guarantees, since its best MDTO tends to be worse than the best "soft_min".

4.3 Recommendations

Based on the results statistics in training and testing, we discuss recommendations when selecting minimal depth sensor representations for adjusting DDQN-based agents. From empirical results, choosing the mean sampling method achieves the most reliable performance for fine-tuning DSRMs, resulting in monotonically improving metrics as more sectors are added. Still, the "soft_min" sampling methods can yield satisfactory results, at the cost of a less intuitive hyperparameter tuning. The most clear trade-off is that using the "mean" sampling method will not guarantee the most optimal safety performance (higher MDTO when compared with "soft_min"), but yields better energy efficiency with similar or identical success and collision rates.

5 Concluding Remarks

Exploring the open problem of mapless autonomous navigation, this work presents, discusses and proposes solutions to objectively adjust state representation based on the final performance expected from the agent. Different sampling intervals and metrics are proposed for depth state representation, using a autonomous ground vehicle as the agent with 2D-LiDAR sensing, and the Double Deep Q-Network model is formally defined and applied to each sensing strategy.

Using a relatively simple neural network architecture and widespread-use methodology, 12 models are trained, tested and made publicly available for usage, as well as the supplementary material for replicating the results and analyses. Further, evidence-based recommendations are proposed in regard of task fulfillment, safety and energy efficiency metrics, introducing an empirical first attempt at finding the minimum optimal depth state representation for mapless autonomous navigation.

Most models were able to converge in training, yet yield different performance in terms of mean and standard deviation of task-related metrics during tests. The "mean" sampling method is shown as the most stable and efficient strategy for fine-tuning state representation, with due regards to safety guarantees.

5.1 Future Works and Suggested Applications

Building upon this work, in terms of methodology, neural network architectures with more hidden layers, different activation functions or involving convolution may yield even better generalization and performance. This also enables tackling different tasks in sensor representation such as sensor fusion (gathering data from multiple different sensors of a single agent) as well as collaborative sensor representation states (gathering sensor data from multiple agents, [24]). Moreover, the problem of minimum optimal representation may be studied similarly to other sampling problems [25].

In addition, the DSRM proposal can be abstracted and generalized by applying Representation Learning, a growing field with excellent performance since the proposal of the attention mechanism [26], which is yet under-explored in Robotics when compared to Natural Language Processing (eg. Large Language Models). Moreover, in terms of applications, adding dynamic obstacles and goals represent an important improvement for a more general solution to the mapless autonomous navigation task in real environments. Another important application is regarding different ground vehicles or even unmanned aerial ones.

Acknowledgments

This work has been supported by the Brazilian Ministry of Science, Technology and Innovation (grant 56/2022), as well as CNPq, FNDTC, CAPES and FAPEMIG.

References

[1] H. Lee and J. Jeong, "Mobile robot path optimization technique based on reinforcement learning algorithm in warehouse environment," *Applied sciences*, vol. 11, no. 3, p. 1209, 2021.

[2] H. Wang and N. Noguchi, "Adaptive turning control for an agricultural robot tractor," *International Journal of Agricultural and Biological Engineering*, vol. 11, no. 6, pp. 113–119, 2018.

[3] X. Liu and Y. Tan, "Feudal latent space exploration for coordinated multi-agent reinforcement learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[4] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[5] J. Clifton and E. Laber, "Q-learning: Theory and applications," *Annual Review of Statistics and Its Application*, vol. 7, pp. 279–301, 2020.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[7] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *Advances in Neural Information Processing Systems*, vol. 35, pp. 507–520, 2022.

[8] M.-F. R. Lee and S. H. Yusuf, "Mobile robot navigation using deep reinforcement learning," *Processes*, vol. 10, no. 12, p. 2748, 2022.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[11] M. S. Güzel, "Autonomous vehicle navigation using vision and mapless strategies: a survey," *Advances in Mechanical Engineering*, vol. 5, p. 234747, 2013.

[12] B. Padmaja, C. V. Moorthy, N. Venkateswarulu, and M. M. Bala, "Exploration of issues, challenges and latest developments in autonomous cars," *Journal of Big Data*, vol. 10, no. 1, p. 61, 2023.

[13] X. Zhang, X. Shi, Z. Zhang, Z. Wang, and L. Zhang, "A ddqn path planning algorithm based on experience classification and multi steps for mobile robots," *Electronics*, vol. 11, no. 14, p. 2120, 2022.

[14] G. Tong, N. Jiang, L. Biyue, Z. Xi, W. Ya, and D. Wenbo, "Uav navigation in high dynamic environments: A deep reinforcement learning approach," *Chinese Journal of Aeronautics*, vol. 34, no. 2, pp. 479–489, 2021.

[15] N. Botteghi, R. Obbink, D. Geijs, M. Poel, B. Sirmacek, C. Brune, A. Mersha, and S. Stramigioli, "Low dimensional state representation learning with reward-shaped priors," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 3736–3743.

[16] L. D. de Moraes, V. A. Kich, A. H. Kolling, J. A. Bottega, R. Steinmetz, E. C. da Silva, R. Grando, A. R. Cuckla, and D. F. T. Gamarra, "Double deep reinforcement learning techniques for low dimensional sensing mapless navigation of terrestrial mobile robots," in *Intelligent Systems Design and Applications*, A. Abraham, S. Pllana, G. Casalino, K. Ma, and A. Bajaj, Eds. Cham: Springer Nature Switzerland, 2023, pp. 156–165.

[17] P. Zhang, C. Wei, B. Cai, and Y. Ouyang, "Mapless navigation for autonomous robots: A deep reinforcement learning approach," in *2019 Chinese Automation Congress (CAC)*. IEEE, 2019, pp. 3141–3146.

[18] K. B. de Carvalho, I. R. L. de Oliveira, and A. S. Brandão, "Av navigation in 3d urban environments with curriculum-based deep reinforcement learning," in *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2023, pp. 1249–1255.

[19] H. Zhang, M. Huang, H. Zhou, X. Wang, N. Wang, and K. Long, "Capacity maximization in ris-uav networks: a ddqn-based trajectory and phase shift optimization approach," *IEEE Transactions on Wireless Communications*, vol. 22, no. 4, pp. 2583–2591, 2022.

[20] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[21] B. J. Heil, M. M. Hoffman, F. Markowetz, S.-I. Lee, C. S. Greene, and S. C. Hicks, "Reproducibility standards for machine learning in the life sciences," *Nature Methods*, vol. 18, no. 10, pp. 1132–1135, 2021.

[22] G. Chen, H. Yu, W. Dong, X. Sheng, X. Zhu, and H. Ding, "What should be the input: Investigating the environment representations in sim-to-real transfer for navigation tasks," *Robotics and Autonomous Systems*, vol. 153, p. 104081, 2022.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] W. Chen, S. Zhou, Z. Pan, H. Zheng, and Y. Liu, "Mapless collaborative navigation for a multi-robot system based on the deep reinforcement learning," *Applied Sciences*, vol. 9, no. 20, p. 4198, 2019.

[25] M. Kohler and S. Langer, "On the rate of convergence of fully connected very deep neural network regression estimates," *arXiv preprint arXiv:1908.11133*, 2019.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.