


Article

Self-Learning Robot Autonomous Navigation with Deep Reinforcement Learning Techniques

Borja Pintos Gómez de las Heras ^{*}, Rafael Martínez-Tomás  and José Manuel Cuadra Troncoso 

Department of Artificial Intelligence, National Distance Education University, Juan del Rosal 16, 28040 Madrid, Spain; rmtomas@dia.uned.es (R.M.-T.); jmcuadra@dia.uned.es (J.M.C.T.)

* Correspondence: bpintosgomezheras@gmail.com

Abstract: Complex and high-computational-cost algorithms are usually the state-of-the-art solution for autonomous driving cases in which non-holonomic robots must be controlled in scenarios with spatial restrictions and interaction with dynamic obstacles while fulfilling at all times safety, comfort, and legal requirements. These highly complex software solutions must cover the high variability of use cases that might appear in traffic conditions, especially when involving scenarios with dynamic obstacles. Reinforcement learning algorithms are seen as a powerful tool in autonomous driving scenarios since the complexity of the algorithm is automatically learned by trial and error with the help of simple reward functions. This paper proposes a methodology to properly define simple reward functions and come up automatically with a complex and successful autonomous driving policy. The proposed methodology has no motion planning module so that the computational power can be limited like in the reactive robotic paradigm. Reactions are learned based on the maximization of the cumulative reward obtained during the learning process. Since the motion is based on the cumulative reward, the proposed algorithm is not bound to any embedded model of the robot and is not being affected by uncertainties of these models or estimators, making it possible to generate trajectories with the consideration of non-holonomic constraints. This paper explains the proposed methodology and discusses the setup of experiments and the results for the validation of the methodology in scenarios with dynamic obstacles. A comparison between the reinforcement learning algorithm and state-of-the-art approaches is also carried out to highlight how the methodology proposed outperforms state-of-the-art algorithms.



Citation: Pintos Gómez de las Heras, B.; Martínez-Tomás, R.; Cuadra Troncoso, J.M. Self-Learning Robot Autonomous Navigation with Deep Reinforcement Learning Techniques.

Appl. Sci. **2024**, *14*, 366. <https://doi.org/10.3390/app14010366>

Academic Editor: Yutaka Ishibashi

Received: 10 November 2023

Revised: 20 December 2023

Accepted: 28 December 2023

Published: 30 December 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: autonomous robots; deep reinforcement learning; dynamic environment; comfort driving; self-learning

1. Introduction

Autonomous driving is a very wide field of research that gathers many different methodologies. These methodologies are usually targeted to a very specific problem within the autonomous driving domain. The methodology presented in this work proposes a solution for a robot driving in a scenario with spatial restrictions (limited drivable space), interacting with static and dynamic obstacles, and fulfilling at all times safety, legal, and comfort requirements. The safety requirements ensure that the robot moves within the drivable space, avoiding collision with any other obstacle and always fulfilling the non-holonomic constraints that the robot might have. The legal requirements define the maximum speed limit and the comfort requirements establish limitations on the robot's acceleration and jerk to maximize comfort. The state-of-the-art solutions that attempt to solve this autonomous driving problem are characterized by overly complicated algorithms that demand a lot of computational resources. This article uses deep reinforcement learning techniques to overcome these issues. Specifically, the proposed methodology contains the following contributions:

- (1) The methodology relies on the definition of simple, multiple, and intuitive reward functions. The reward functions are closely related to the feeling of pain or pleasure that we might receive after performing a series of actions. In contrast to a cost function, the reward function does not evaluate how good or bad it is to take a specific action in a given state.
- (2) The perception, motion planning, and motion control software modules are not embedded in a complex deep neural network architecture. Only the motion planning software module is implemented using a deep neural network model, making training easier.
- (3) A continuous multi-action space is defined to solve complex dynamic scenarios.
- (4) The comfort requirements are easily included through the definition of simple reward functions. Since all reward functions are normalized between -1 and 0 , a weighted sum of the reward functions is performed to prioritize safety requirements over legal or comfort requirements.

The rest of the paper is structured as follows. The Materials and Methods section reviews the relevant literature and gives a detailed explanation of the proposed methodology. The design of experiments for validation of the methodology and the simulation results are also discussed in the Discussion section and compared with state-of-the-art algorithms. Finally, the main conclusions and advantages of the proposed methodology are listed in the Conclusions section.

2. Materials and Methods

To review the related work, we went through all robotic paradigms, always keeping in mind the autonomous driving problem that we were attempting to solve. The definition of this autonomous driving problem has basically 4 challenging points: first, the accomplishment of the safety and legal requirements under dynamic scenarios, i.e., avoiding dynamic obstacles and driving the robot within the drivable space, controlling its maximum linear speed. Some methods in the literature struggle to accomplish these requirements under dynamic scenarios. Second, the fulfillment of comfort requirements. Some robotic paradigms are less suitable than others to fulfill comfort requirements. Third, the generation of feasible trajectories, i.e., the generation of trajectories that are capable of being realized by the motion control. This point represents a headache for many state-of-the-art algorithms, specifically when facing driving situations close to the limit (grip limit). Finally, the proposed algorithm must have a reasonable computational load. State-of-the-art algorithms are well known for demanding very high computational resources, making them hard to run on microcontrollers.

A. Reactive Paradigm

Methods based on the reactive paradigm perform well with holonomic robots in spatially restricted scenarios with static obstacles while fulfilling safety and legal requirements. Common methods are the potential field method [1], velocity obstacle method [2], dynamic window approach [3], or partial center of area method [4]. However, non-holonomic robots might be difficult to integrate with the reactive paradigm since they need to plan their movement ahead in order to avoid collision with obstacles. Moreover, the control based on reactions to certain sensorial inputs (behaviors) without any local planning can lead to aggressive and long trajectories [5], making it difficult to integrate comfort requirements. On the other hand, the reactive paradigm demands low computational resources, which enables the usage of non-expensive microcontrollers.

B. Deliberative Paradigm

The robotic paradigm with a local planification stage is the deliberative paradigm. The state-of-the-art solutions that attempt to solve the previously described autonomous driving problem are mainly grouped in this robotic paradigm. The methods within the deliberative paradigm can be divided into three different subgroups: graph-based methods, sampling-based methods, and model predictive control methods.

Within the graph-based methods group, the Dijkstra algorithm [6] performs very well at avoiding obstacles and finding the shortest path. However, it is difficult to use with large grids because it is computationally very expensive. The A* algorithm [7] followed the Dijkstra algorithm to reduce the computational load by including heuristic functions to find the solution faster. However, the Dijkstra and A* algorithms have also an additional and very important disadvantage: they do not consider the robot dynamics in the motion planning, making the method invalid for non-holonomic robots. Kinematic or dynamic embedded models can be considered to make the method valid for non-holonomic robots. A good example is the hybrid A* algorithm [8]. However, these methods rely on embedded models that are affected by uncertainties, producing trajectories that might be unfeasible to control, specifically if the robot faces driving situations close to the limit.

With the aim of reducing the computational cost, sampling-based methods were created. Sampling-based methods can work on a continuous space where only specific discrete waypoints are sampled. Common methods are rapidly exploring random trees (RRTs) [9] or the probabilistic roadmap method (PRM) [10]. Once the sampling process is completed, a path considering non-holonomic constraints can be generated: [11] used maximum and minimum limitations of a path's curvature and the momentum of the vehicle and [12] used an embedded dynamic model of the robot. Eventually, the final trajectory was selected based on the score given by a cost function. Since non-holonomic constraints were considered, the cost function only needed to evaluate aspects such as robot collisions, comfort, or total travel time. However, [11] did not consider all non-holonomic constraints and [12] relied on embedded models that could differ from reality. Driving situations close to the limit are also challenging, even when using complex embedded robot dynamic models. Feasible trajectories can be easily generated under scenarios of low velocity, such as parking [13] or scenarios with moderate accelerations [14]. But, scenarios close to the limit are difficult to consider in this approach. Some other approaches based on pure mathematical functions can be applied to connect the sampled waypoints with a smooth path profile, such as Bezier curves [15], splines curves [16], or the lattice planner with polynomial curves for the path's curvature [17]. These approaches can have the advantage of not embedding a dynamic model of the robot into the logic, which can simplify the implementation. However, since some of the trajectories generated might not fulfill the non-holonomic constraints, the cost function must consider these constraints to reject non-feasible trajectories.

The predictive control model attempts to generate feasible trajectories considering all non-holonomic constraints, without having to completely discretize (graph-based) or partially sample (sampling-based) the continuous space. An embedded dynamic model of the robot is included in the logic. A cost function defines the goals of the autonomous driving problem and an optimizer iteratively solves this problem, minimizing the cost function and generating the corresponding control actions over a future horizon. The predictive control model also addresses the problem of planning feasible trajectories under driving situations close to the limit. In [18], the parameters of the embedded model were updated over the horizon in situations close to the limit based on a predictive friction estimate to obtain even more reliable and feasible trajectories. However, there was still a reliance on models or estimators that could have differences with respect to reality and the computational consumption was still very high due to the usage of an iterative optimizer.

In general, the main problems of the deliberative paradigm are the challenging design of the cost function, the high computational cost, and the uncertainties of embedded dynamic models, no matter whether the algorithm falls under a graph-based, sampling-based, or predictive control model. In [8], the challenges of designing an effective cost function for the hybrid A* algorithm were addressed. In [19], a comparison of motion planning algorithms was performed in terms of computational cost, highlighting the high computational cost of graph-based and sampling-based methods.

C. Sense–Think–Act Paradigm

The problem of the uncertainties of embedded models, which eventually compromises the feasibility of the generated trajectory, is addressed by the robotic paradigm sense–think–act. In this paradigm, a model, usually a deep neural network, is trained with real trajectories (supervised learning) to create a path planner algorithm based on artificial intelligence [20]. Thus, it avoids the usage of embedded dynamic models. However, the main drawbacks of this paradigm are the need for big databases and the labeling of these databases. The labeling of databases is not only a very time-consuming task, but it can also introduce bias to the results, specifically if the labeling task is done manually.

D. Sense–Explore–Act–Learn Paradigm

The methods based on reinforcement learning techniques attempt to bring a solution to the previously mentioned drawbacks. The drawbacks and the solution proposed by the reinforcement learning approach are summarized in Table 1.

Table 1. Strengths of reinforcement learning-based algorithms for autonomous driving navigation.

State-of-the-Art Drawbacks	Reinforcement Learning Strengths
Cost function design	Cost function is automatically learned
Performance under dynamic environments	Agent learns from dynamic environments
Uncertainties of embedded dynamic models	Agent free from embedded dynamic models
High computational cost	Low computational cost avoiding iterative motion planning modules
Integration of comfort requirements	Easy integration with simple reward functions
Databases must be available and labeled	Data are generated on-the-go and automatically labeled based on a reward value

Due to the advantages summarized in the previous table, reinforcement learning techniques have been the subject of extensive study within the autonomous driving domain. In [21], Q-learning was used to find the shortest path from the starting pose to the destination pose. Reinforcement learning techniques with continuous actions and states were introduced in [22], with very promising results. A deep deterministic policy gradient (DDPG) was introduced in [23] to learn from a real scenario how to drive and keep the vehicle inside the road boundaries by providing just a single camera image of the road as the input. However, this work only implemented partial safety requirements (driving within the drivable space without obstacle avoidance) and did not consider any comfort or legal requirements. Additionally, the perception, motion planning, and motion control modules were embedded in the deep neural networks of the deep reinforcement learning algorithm. In [24], the autonomous driving algorithm relied not only on input images of the cameras as in [23] but also on post-processed data as velocities or relative distances. In [25], comfort requirements were included for lane change maneuvers as a simple reward function. In contrast to [23], it separated the perception module from the reinforcement learning task, and the agent consumed already post-processed signals from the perception module, such as robot velocities or relative distances, instead of images. The algorithm presented in this work keeps the perception, motion planning, and motion control modules separated from each other. Images are not directly provided to the agent. Instead, all the relevant post-processed information from the perception module is provided to the agent so that it can learn the best possible trajectory based only on variables that have an important impact on motion planning. Regarding the definition of reward functions, [26] included positively and negatively weighted rewards. The methodology proposed only included negatively weighted rewards to exclude policies that end up in endless loops of collecting permanently positive rewards instead of reaching the destination. In terms of requirements, ref. [26] included vehicle speed (legal requirements), collisions (safety requirements), and travel distance towards the destination (task-oriented requirements), but comfort requirements

were not included. In [27], comfort, legal, and task-oriented requirements (lane keeping) were included, but safety requirements were not, which are needed for, for example, overtaking maneuvers in dynamic scenarios. In [28], all requirements were included. However, the comfort requirements were not expressed in terms of acceleration and jerk but in terms of limiting the steering wheel angle (thus only penalizing lateral acceleration but excluding lateral jerk and longitudinal acceleration and jerk). In [28], a mix of sparse and continuous reward functions was used. The methodology proposed only used continuous reward functions to better guide the reinforcement learning algorithm to find the most valuable rewards.

The methodology proposed is based on the deep deterministic policy gradient (DDPG) algorithm. The DDPG method, like the Q-learning method, is a model-free algorithm. That is, it does not use any embedded model of the robot inside the agent logic. Instead, it learns the optimal policy directly from the environment. The DDPG method is intended for environments with continuous states and actions. This is possible by approximating the actor model (which computes the action a_t) with deep neural network μ of weights θ as a function of state s at time t (1).

$$a_t = \mu(s_t|\theta) = \mu_\theta(s_t) \quad (1)$$

The DDPG algorithm is an actor–critic method and, therefore, the critic function is also approximated with another deep neural network Q of weight ϕ as a function of state s and the action a at time t (2).

$$Q_\phi(s_t, a_t) = Q_\phi(s_t, \mu_\theta(s_t)) \quad (2)$$

Actor–critic methods combine the benefits of policy-based and value-based methods. They benefit from the good convergence properties of policy-based methods and they also benefit from the sample efficiency of value-based methods, which tend to find an optimal solution faster.

The deterministic policy gradient theorem (3) updates the weight θ of the actor model μ_θ in the direction of the maximum value of the critic model Q_ϕ :

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_a Q_\phi(s, a) \Big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s) \right] \quad (3)$$

On the other hand, the Bellman equation is used to iteratively update the weight ϕ of the critic network Q_ϕ (4) and eventually learn the optimal critic and actor networks.

$$Q_\phi(s_t, a_t) = \mathbb{E}_{s \sim \mathcal{D}} \left[r(s_t, a_t) + \gamma Q_\phi(s_{t+1}, \mu_\theta(s_{t+1})) \right] \quad (4)$$

Additionally, this actor–critic method uses the replay buffer and target networks to bring more stability to the learning process. Random observations are selected from the replay buffer to avoid training the neural networks with consecutive and correlated observations. The weights ϕ' and θ' of the target networks $\mu'_{\theta'}(s)$ and $Q'_{\phi'}(s, a)$ are updated after the weights of the actor and critic networks with a soft factor τ (5). This lag brings stability to the learning.

$$\begin{aligned} \theta' &= \tau\theta + (1 - \tau)\theta' \\ \phi' &= \tau\phi + (1 - \tau)\phi' \end{aligned} \quad (5)$$

The neural network architecture of actor and critic models considers several hidden layers (Figure 1). The variables in brackets denote the number of neurons for that layer.

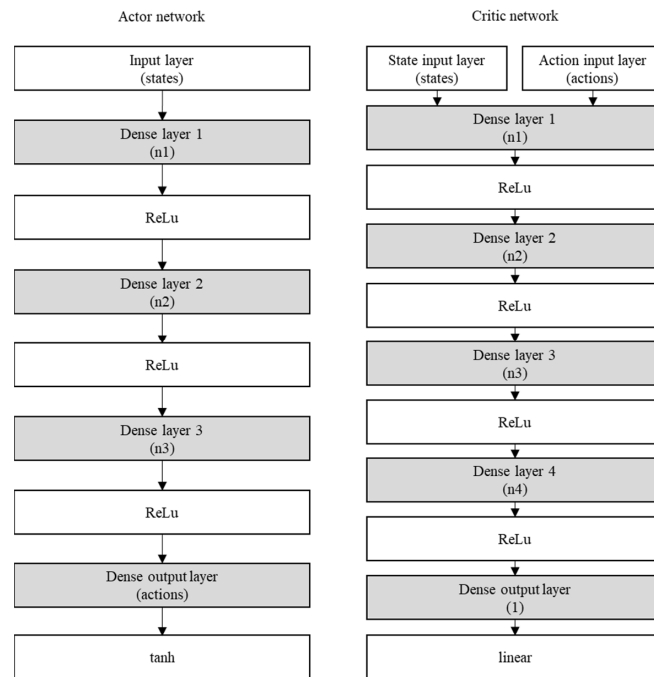


Figure 1. Actor and critic deep neural network architecture.

Finally, the exploration is essential to find the optimal policy. For that purpose, a white noise signal of zero mean value is added to the actor–network output (6). The amplitude of the exploration is controlled with the variance σ of the normal distribution \mathcal{N} .

$$a_t = \mu_\theta(s_t) + \mathcal{N}(0, \sigma) \tag{6}$$

As the learning process progresses and the actor and critic networks learn the optimal values, less exploration is needed. Therefore, a decay of the exploration was programmed. This decay also helped to stabilize the learning process. The DDPG pseudocode is shown in Algorithm 1:

Algorithm 1. Deep Deterministic Policy Gradient

```

Init  $\mu_\theta(s)$  and  $\mu'_{\theta'}(s)$  with weights  $\theta$  and  $\theta' = \theta$ 
Init  $Q_\phi(s, a)$  and  $Q'_{\phi'}(s, a)$  with weights  $\phi$  and  $\phi' = \phi$ 
Init discount factor  $\gamma$  and soft update  $\tau$ 
for episode  $\in \{1 \dots episode_{max}\}$  do
   $s_t = s_0$ 
  while True
     $a_t = \mu_\theta(s_t) + \mathcal{N}(0, \sigma)$ 
    Execute  $a_t$  and observe next state  $s_{t+1}$ , reward  $r_t$ ,
    and termination flag  $T_t$ 
    Record transition  $(s_t, a_t, r_t, s_{t+1})$  in the buffer
    Sample a batch of transitions  $\mathcal{D}$  from the buffer
    Compute  $y_i = r_i + \gamma Q'_{\phi'}(s_{i+1}, \mu'_{\theta'}(s_{i+1}))$ 
    Update  $\phi$  by minimizing the loss function  $L$ :
      
$$L = \frac{1}{N} \sum_{i \in \mathcal{D}} (y_i - Q_\phi(s_i, \mu_\theta(s_i)))^2$$

    Update  $\theta$ :
      
$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s \sim \mathcal{D}} \left[ \nabla_a Q_\phi(s, a) \Big|_{a=\mu_\theta(s)} \nabla_\theta \mu_\theta(s) \right]$$

    Update  $\phi'$  and  $\theta'$ :
      
$$\theta' = \tau\theta + (1 - \tau)\theta'$$

      
$$\phi' = \tau\phi + (1 - \tau)\phi'$$

  if  $T_t == \text{True}$ 
    break

```

Before reaching the maximum number of learning episodes, the learning process can be terminated if some specific conditions are met. When reaching these conditions, it is considered that the policy is mature enough to fulfill the requirements. There are mainly 2 conditions: first, the number of total episodes must be greater than the minimum number. With this condition, we ensured that the exploration rate decayed until almost 0 to avoid very noisy and unstable policies. Secondly, no termination due to non-compliance of the safety, legal, or comfort requirements must be detected for a minimum number of episodes to ensure proper learning. In other words, the termination of the learning episode must be only related to task-oriented requirements, such as if the robot reaches the destination pose. If the episode gets terminated because the robot crashes into an obstacle, the counter to consider that the policy is mature enough gets reset to 0.

The agent, which is based on the DDPG algorithm, interacts with the environment. Important concepts of the environment are the state variables, reward functions, and motion control.

A. State Variables

The state variables S are essential during the reinforcement learning process (7). One important property is that the state variables must be capable of generalization so that the policy obtained after the learning process can solve unseen scenarios. Therefore, absolute variables like the absolute distance traveled by the robot are avoided. A set of relative distances (d_1, \dots, d_n) with respect to the robot's position, the relative orientation to the destination (θ_{dest}), and dynamic variables such as robot linear and angular velocities (v_{rob}, w_{rob}) and accelerations ($\dot{v}_{rob}, \dot{w}_{rob}$) are selected. The relative distances come from a lidar sensor, which covers 180 degrees of the robot's front view. To smoothly reduce the linear velocity before reaching the destination, the relative distance between the robot and the destination (d_{dest}) is also included.

$$S = \{d_1, \dots, d_n, v_{rob}, w_{rob}, \dot{v}_{rob}, \dot{w}_{rob}, \theta_{dest}, d_{dest}\} \quad (7)$$

Figure 2 shows an image of the previously mentioned state variables:

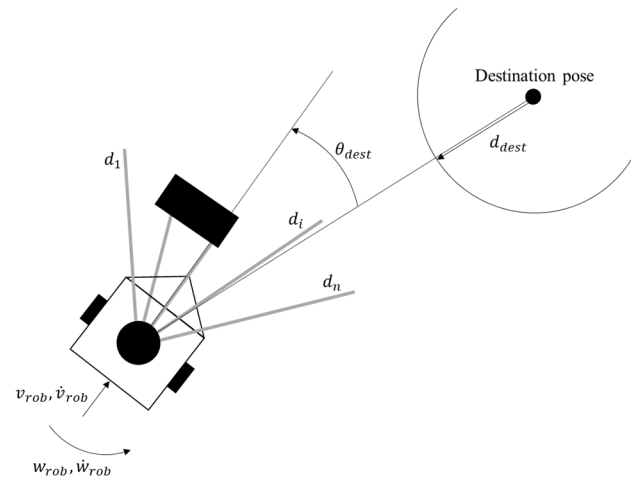


Figure 2. State variables.

For the learning process to be stable, the state variables must fulfill the Markov property. The Markov property is formulated in (8):

$$p(s_{t+1}|s_t) = p(s_{t+1}|s_1, s_2, \dots, s_t) \quad (8)$$

According to (8), the future state s_{t+1} only depends on the current state s_t . This means that all the information from the history (s_1, s_2, \dots, s_{t-1}) is irrelevant for computing the future state s_{t+1} . To make it possible for the state variables to fulfill the Markov property,

they must include not only relative distances but also dynamic variables like the linear and angular velocities and accelerations. Therefore, they are included in the list of state variables in (7).

Finally, it is necessary to make sure that all states variables can be measured with sensors. The relative distances come from a lidar sensor mounted on the top of the robot, covering 180 degrees of the front view. The linear and angular velocities and accelerations of the robot can be measured with encoders mounted on the driven wheels and IMU sensors. The distance and orientation to the destination pose can be calculated based on camera images. The destination pose can be recognized by using some special color that the camera recognizes. Based on that recognition and after proper calibration of the camera parameters, the distance to the destination pose and the relative orientation of the robot to the destination pose can be calculated.

B. Reward Functions

The reward functions are the key to the reinforcement learning algorithm. The reward function does not define which action to take. Instead, the reward functions are a measurement of how good or bad it is to end up in a new state after an action is taken. This measurement of good or bad is closely related to the sensation of pain or pleasure. Based on the reward functions, the reinforcement learning algorithm automatically learns a complex critic function to navigate the robot autonomously. In contrast to the reward function, the critic function defines, for a given state, how good an action is, before the robot performs such an action. Comparing the critic function with the state-of-the-art algorithms like the lattice planner or the predictive control model, the critic function is equivalent to the cost function. In these state-of-the-art algorithms, the cost function is usually a single and very complex function, which needs to be manually designed to fulfill the requirements of the autonomous driving problem. On the other hand, the reinforcement learning algorithm automatically learns this cost function (critic function) based on simple, multiple, and very intuitive reward functions. Therefore, the critic function gathers all the experience seen during the learning process to build a complex function that fulfills all requirements. To learn this complexity, usually, deep neural networks are selected as policy and critic approximation functions.

There are some important rules to follow before the definition of the reward functions:

- In problems with continuous actions and states, or in case of too large discrete spaces, it is very convenient to shape the reward functions instead of using sparse rewards. The reason behind this is that the robot learns by means of exploration. If the robot starts randomly exploring, the probability of the robot seeing this sparse reward can be very low, leading to either an unsuccessful learning process because the robot does not find the reward or to an extremely long learning process. On the other hand, in case of shaped rewards (rewards with a smooth continuous gradient), the robot can adapt its actions in the direction of maximizing the cumulative reward right from the beginning.
- It is better to avoid purely negative rewards. If the reward functions are always negative, excluding the 0, the reinforcement learning might find a solution by terminating the learning episode as soon as possible rather than keeping on driving and therefore accumulating more negative rewards.
- Positive rewards are also tricky, and they must be treated carefully. If the robot finds a positive reward, the logic might end up in a loop solution.
- The learning scenarios must always be carefully designed to facilitate finding rewards from a beneficial starting situation.

Regarding the type of rewards, they are divided into safety, legal, comfort, and task-oriented requirements. The safety, legal, and comfort requirements are based on the pain that the robot might suffer if they are not fulfilled. On the other hand, task-oriented requirements are based on the sensation of pleasure that is obtained by fulfilling the

proposed tasks. Therefore, depending on the robot's mission, task-oriented requirements can be built following different criteria.

(1) Safety requirements

Safety requirements implement a penalization or negative reward if the robot crashes into obstacles. Situations where the robot drives too close to an obstacle are also penalized even when no contact between the robot and the obstacle happens. For that reason, a safety margin around the obstacle footprint of distance d_{limit} is defined, as represented in Figure 3. A very simple reward function (9) is therefore defined: 0 if the robot drives outside the safety margin, and a linear interpolation between 0 and -1 if the robot drives inside the safety margin. The variable d is the relative distance from the robot to the obstacle.

$$r_{safety} = \begin{cases} 0 & , d > d_{limit} \\ \frac{d-d_{limit}}{d_{limit}} & , d \leq d_{limit} \end{cases} \quad (9)$$

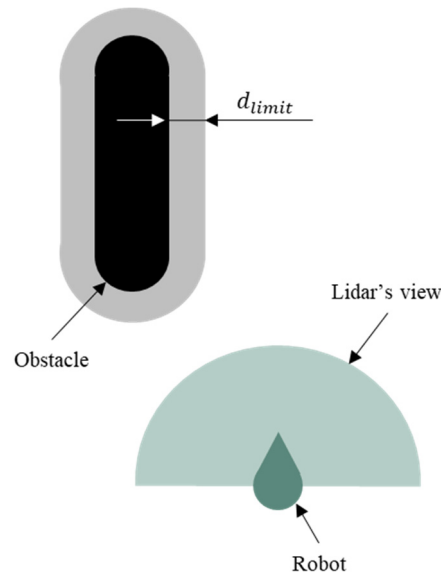


Figure 3. Safety margin around the obstacle footprint. If the robot enters the area delimited by the safety margin, a negative reward will be forwarded to the agent during the learning process.

(2) Legal requirements

Legal requirements limit the maximum speed that the robot can travel. The robot must control its linear speed v_{rob} so that it does not exceed this maximum value. The implementation of this reward function is very simple as well (10). There are two speed thresholds (v_1 and v_2). If the speed does not exceed the lower threshold v_1 , the reward will be 0. The reward will be linearly interpolated between 0 and -1 between the first and second thresholds.

$$r_{legal} = \begin{cases} 0 & , v_{rob} < v_1 \\ -\frac{v_{rob}-v_1}{v_2-v_1} & , v_1 \leq v_{rob} \leq v_2 \\ -1 & , v_{rob} > v_2 \end{cases} \quad (10)$$

If the legal maximum velocity is v_{legal} , v_1 can be equal to v_{legal} and v_2 can be set a little bit higher than v_{legal} .

(3) Comfort requirements

Comfort can be defined as the pleasant feeling of the people traveling inside the robot. Feelings of discomfort can come from different sources. The three main sources that can decrease the feeling of comfort are listed below:

- High acceleration values (longitudinal and lateral/angular)
- High jerk values (longitudinal and lateral/angular)
- Low-frequency acceleration over long periods of time (feeling of sickness)

In this work, the first source of discomfort is implemented. The reward function (11) defines a window given two acceleration values, \dot{w}_1 and \dot{w}_2 . Accelerations below \dot{w}_1 are not penalized. Accelerations within \dot{w}_1 and \dot{w}_2 are penalized using linear interpolation. The maximum penalization value is clipped to -1 if the acceleration is higher than \dot{w}_2 .

$$r_{comfort} = \begin{cases} 0 & , \dot{w}_{rob} < \dot{w}_1 \\ -\frac{\dot{w}_{rob}-\dot{w}_1}{\dot{w}_2-\dot{w}_1} & , \dot{w}_1 \leq \dot{w}_{rob} \leq \dot{w}_2 \\ -1 & , \dot{w}_{rob} > \dot{w}_2 \end{cases} \quad (11)$$

Reward functions for longitudinal and angular accelerations are added together to build the final comfort function.

(4) Task-oriented requirements

It is possible to define multiple and specific tasks to be accomplished during autonomous navigation. In this work, the robot is intended to reach a specific destination pose. For that purpose, the robot is rewarded when facing the right direction towards the destination pose. The robot will be also rewarded when traveling at the maximum legal speed.

For the first task, if the direction to the destination pose θ_{target} (determined by visual sensors) is not the same as the robot's heading θ_{rob} , the robot is penalized according to (12).

$$r_{destination} = -\frac{|\theta_{target} - \theta_{rob}|}{\pi} \quad (12)$$

If the destination cannot be detected by visual sensors, the destination reward is set to 0 and the robot moves forward within the drivable space.

For the second task, the robot is penalized if the linear speed is not the maximum legal one (13). Since legal requirements are also implemented, if the robot exceeds the maximum legal speed, it is penalized according to (10).

$$r_{speed} = \begin{cases} -\frac{v_{target}-v_{rob}}{v_{target}} & , v_{rob} < v_{target} \\ 0 & , v_{rob} \geq v_{target} \end{cases} \quad (13)$$

In the previous reward function, the target linear velocity v_{target} is equal to v_{legal} .

Finally, the reward due to the task-oriented requirements is obtained by adding the single functions (14):

$$r_{task} = r_{destination} + r_{speed} \quad (14)$$

(5) Final reward function

The final reward function can be simply built by adding the single reward functions (15). Each reward function is normalized between -1 and 0. A weighted addition of the single reward functions is performed to prioritize the requirements:

$$r = K_{safety} * r_{safety} + K_{legal} * r_{legal} + K_{comfort} * r_{comfort} + K_{task} * r_{task} \quad (15)$$

In general, the order of importance within the requirements is as follows: safety, legal, comfort, and task-oriented (16):

$$K_{safety} \geq K_{legal} \geq K_{comfort} \geq K_{task} \tag{16}$$

C. Motion Control

The motion control strategy depends on the type of robot to be controlled. The commanded signals from the agent are the linear and angular velocities. The motion control takes care of the actuator control to fulfill the commanded velocities. In this work, a differential robot was used for the validation. Considering that a differential robot traveling at a constant linear speed must fulfill non-holonomic constraints as well, the differential robot is a simple but valid approach to validate the methodology.

The commanded linear and angular velocities of the robot were converted in wheel rotational speed by means of the kinematic equations of the differential robot (17) and (18).

$$w_{left} = \frac{2v_{rob} + w_{rob}D}{2r_{left}} \tag{17}$$

$$w_{right} = \frac{2v_{rob} - w_{rob}D}{2r_{right}} \tag{18}$$

The differential robot geometry and kinematic variables are shown in Figure 4:

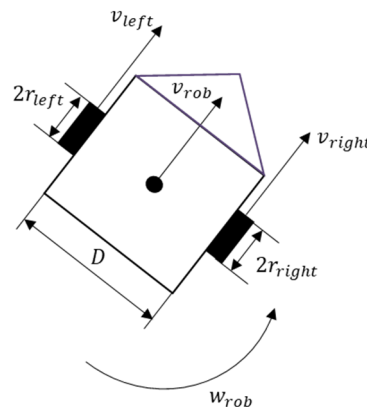


Figure 4. Kinematic variables for a differential robot.

Finally, two independent PI controllers control the commanded velocities by adapting the duty cycles of the electrical motors connected to the driven wheels. Figure 5 shows the final cascade control of the robot’s motion. An external motion planning loop based on deep reinforcement learning techniques generates the requested linear and angular speeds of the robot, depending on the robot’s state measured from the environment. These requested linear and angular speeds are forwarded to an internal control loop based on classical PID controllers. The internal loop measures the linear and angular actual speeds of the robot for closed-loop control.

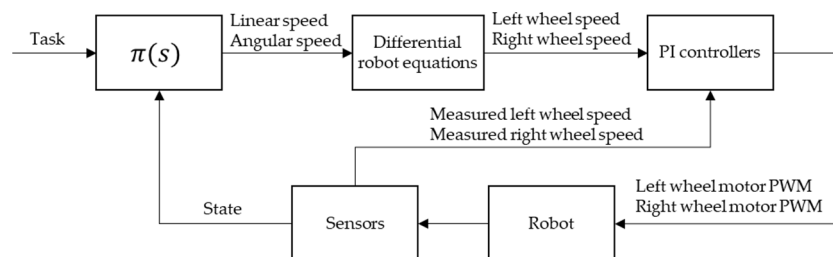


Figure 5. Motion control diagram. Cascade control for the position of the robot.

3. Results

All the experiments were carried out in the simulator CoppeliaSim (Figure 6). A Python script controlled the connection with the simulation by means of APIs. The main libraries used to build the reinforcement learning algorithm were TensorFlow and Gym. Tensorflow and Keras provide the necessary functions to build actor and critic neural network models. Gym provides the necessary tools to implement and compare reinforcement learning algorithms. The actor neural network (policy) produced linear and angular speed set points, which were converted into left and right wheel speeds of the differential robot. These wheel speeds were sent over to the simulator via APIs. The simulator implemented PID controllers to control in a closed loop these speeds.

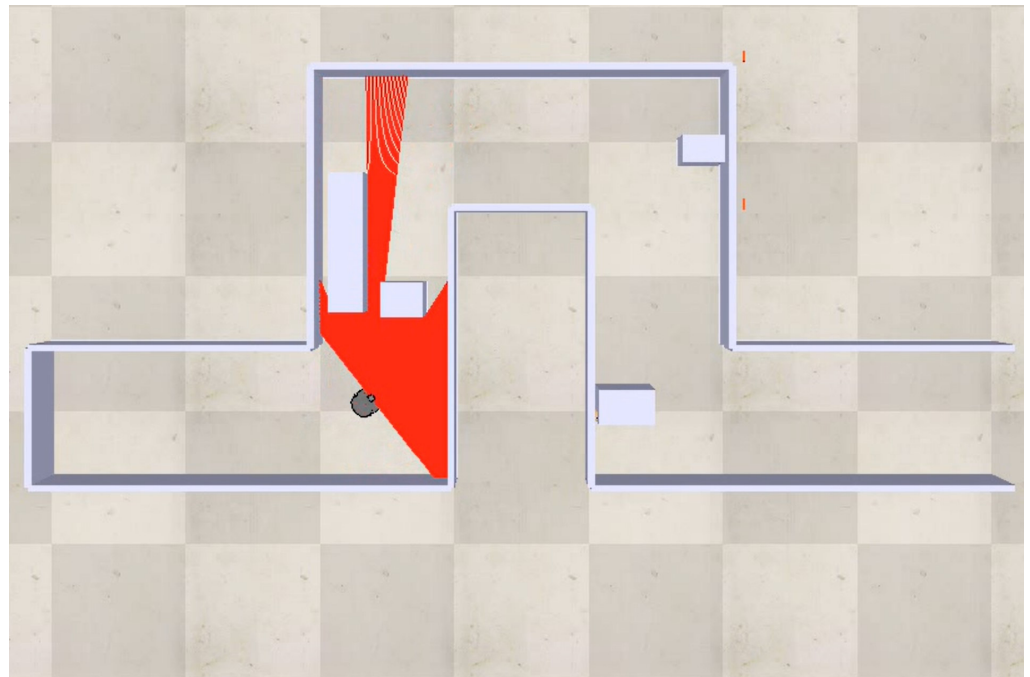


Figure 6. CoppeliaSim simulation platform where all experiments were carried out.

The experiments defined within this section can be divided into two main groups. First, an experiment or set of experiments used for the learning process. Second, an experiment or set of experiments used for the validation process. It was important to use different experiments for the validation process to prove that the methodology proposed can be generalized to new unseen scenarios after the learning process is completed.

For the validation purpose, an experiment including a complex scenario with dynamic and static obstacles was set up. The scenario (see Figure 7) was carefully chosen to validate the following maneuvers:

- (1) Follow-up
- (2) Overtaking
- (3) Lane keeping
- (4) Lane change

Before the robot could perform in this validation experiment, it needed to learn how to drive autonomously. For the learning purpose, less complex experiments were set up. In this case, it was important to carefully select the experiments and scenarios so that the robot could easily find rewards and learn a policy that would enable the robot to generalize and drive autonomously in other unseen and more complex scenarios. Therefore, the learning scenario (Figure 8) was specially adapted to see reward values early in the learning process and better guide the policy to learn the right strategy.

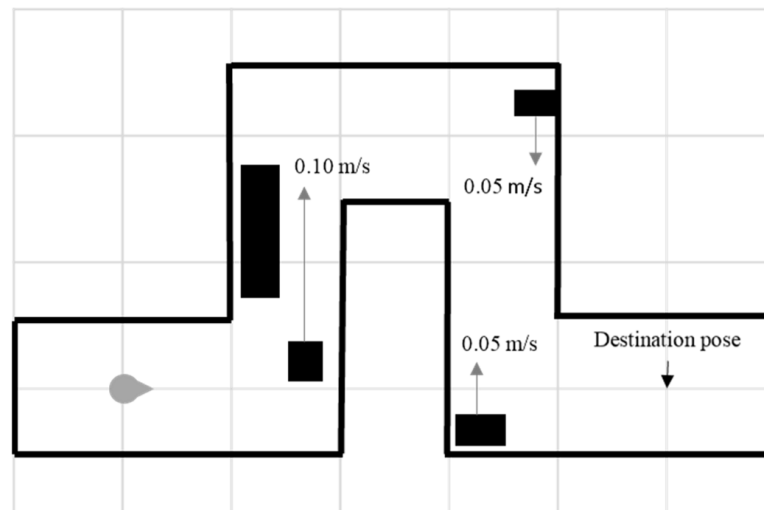


Figure 7. Validation scenario. First dynamic obstacle blocked the way, forcing the robot to follow up the obstacle. Second and third dynamic obstacles were driving at opposite speeds. The robot was forced to perform lane change maneuvers to overtake the obstacles.

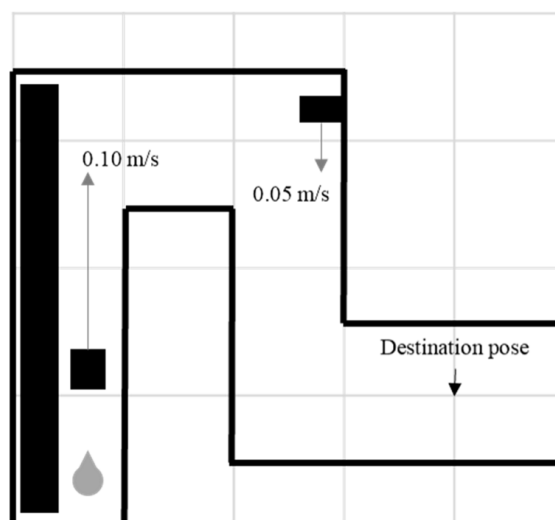


Figure 8. Learning scenario. Left static obstacle reduced the drivable space so that the robot received rewards right from the beginning of the learning process.

The static obstacle of the left hand reduced the drivable space of the robot. This allowed the robot to receive a negative reward right from the beginning if it got too close to the walls, which delimited the boundaries of the circuit. Additionally, the starting pose of the robot was not too far away from the first dynamic obstacle. This helped the robot to quickly see negative rewards while approaching the dynamic obstacle and adapt the policy and critic functions to follow up the obstacle instead of crashing into it. The resulting robot's trajectory after the learning process is shown in Figure 9. The robot's linear velocity was reduced to follow up the dynamic obstacle at the beginning of the circuit and the commanded angular velocity was always smooth to maximize comfort (Figure 10).

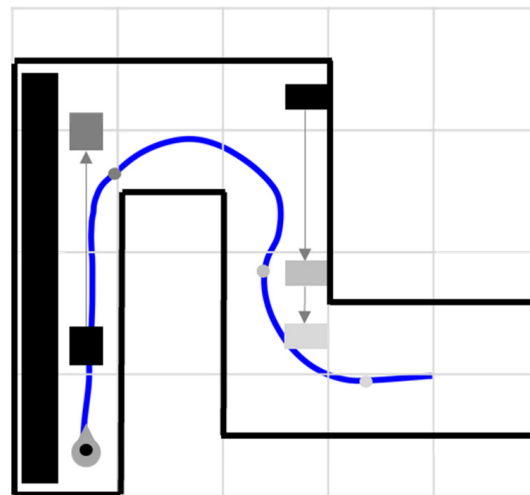


Figure 9. Traveled path by the robot in the learning scenario. Same colors indicate instant positions of the dynamic obstacles and the robot in the traveled path.

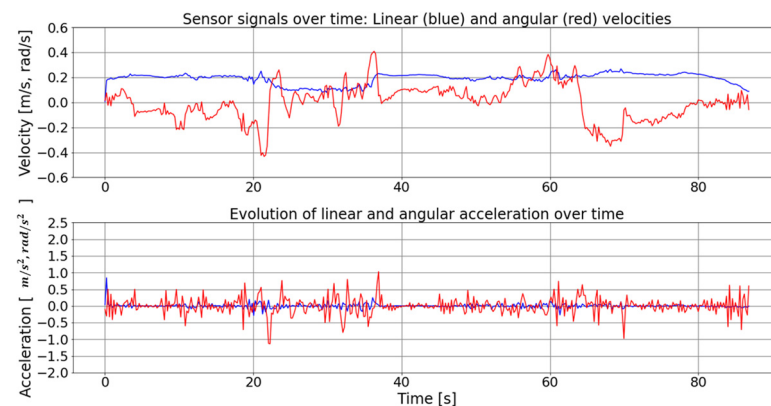


Figure 10. Top graph: linear (blue) and angular (red) velocities of the robot in the learning scenario. Bottom graph: linear (blue) and angular (red) accelerations. The maximum and minimum levels of the linear and angular accelerations were limited to maximize comfort.

Figure 11 shows the episodic reward as a function of the learning episodes. The episodic reward increased towards 0 after each learning episode. There was even room for improvement as the episodic reward did not seem to be completely flat after more than 200 learning episodes. It is interesting to note the recovery process in the episodic reward plot, that is, the episodic reward suddenly went down and then up again in the next episodes. This was due to two different factors: first, the exploration ratio sometimes led the robot to collide with the obstacles. In the beginning, when the policy was still immature and the exploration ratio was high, the recovery peaks could be seen more frequently. Second, when updating the policy following the gradient descent approach, the step or learning rate could be too large, producing a policy that crashes the robot into the obstacles. For these two reasons, a decay for both exploration and learning rate was implemented. This is the reason why as the learning process went on, the recovery peaks were less frequent. The total time needed to automatically learn the policy was 6.57 h. The system used for the learning process was a desktop computer with an AMD Ryzen 5 2600 Six-Core Processor of 3.40 GHz and 8 Gb RAM installed (no GPU installation).

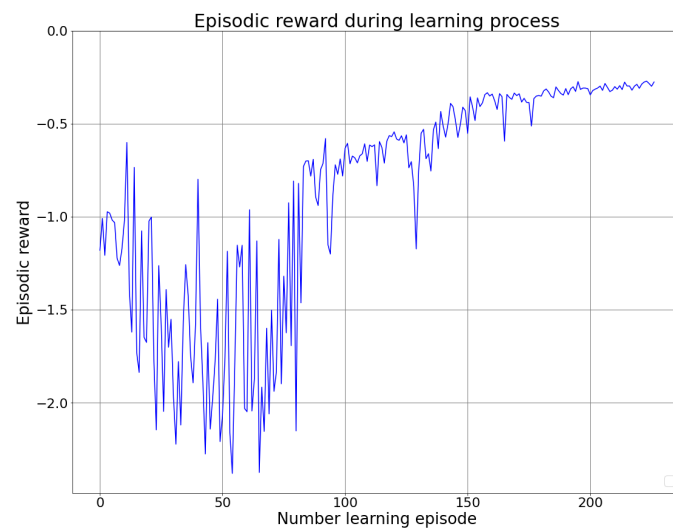


Figure 11. Episodic reward versus learning episodes of the learning scenario.

Finally, the traveled path of the robot in the validation scenario is shown in Figure 12. The first dynamic obstacle blocked the way, and the robot needed to adapt its linear speed to follow up the obstacle. Once the robot detected enough free space, it overtook the obstacle. The second and third obstacles traveled at opposite speeds. Lane change maneuvers were needed to overtake both obstacles. The linear and angular velocities and accelerations are shown in Figure 13. The robot safely avoided all static and dynamic obstacles of the circuit traveling at the maximum legal linear speed of 0.2 m/s (except in the follow-up sector). In addition, the longitudinal and angular accelerations did not exceed the maximum comfort levels defined for this scenario (0.5 m/s^2 and $1 \frac{\text{rad}}{\text{s}^2}$, respectively). Finally, the robot reached the destination pose and ramped down the linear velocity to a standstill position. All of this was happening in a scenario never used during a learning process, concluding that the policy learned during the learning process can be generalized to new and unseen scenarios.

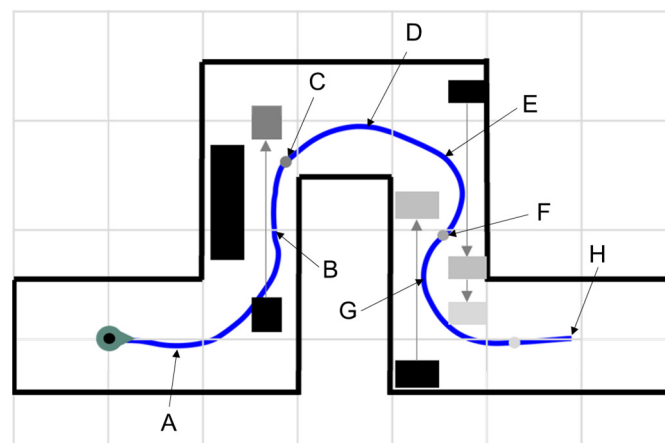


Figure 12. Traveled path by the robot in the validation scenario. Same colors indicate instant positions of the dynamic obstacles and the robot in the traveled path. A, D and G marks indicate a preparation of the robot before the curve to maximize comfort feeling. B mark is the follow up section. C, E and F marks show the moment when the robot found enough free space to overtake. Finally, mark H is the destination pose.

There are several key points marked with capital letters in Figures 12 and 13. These points are explained in detail in Table 2.

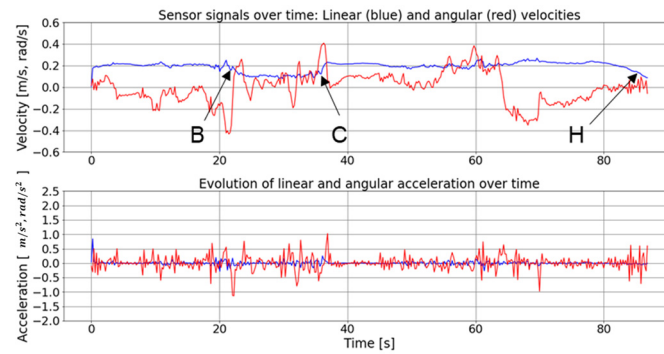


Figure 13. Top graph: linear (blue) and angular (red) velocities of the robot in the validation scenario. Bottom graph: linear (blue) and angular (red) accelerations. The maximum and minimum levels of the linear and angular accelerations were limited to maximize comfort. Between marks B and C is the follow up section, where the robot slowed down its linear speed. Mark H shows the stopping procedure.

Table 2. Explanation of the key points of the traveled path by the robot in the validation scenario.

Mark	Description
A	The robot drove outside of the track before entering the curve to reduce angular acceleration and maximize comfort.
B	The robot followed up the dynamic obstacle at a constant linear speed, keeping a fixed distance between the obstacle and the robot.
C	The robot detected enough free space to overtake the dynamic obstacle and increased its linear speed up to the legal one.
D	Similar to A, the robot drove outside of the track before entering the curve to reduce angular acceleration.
E	The robot detected free space and drove to the left (lane change) to overtake the second dynamic obstacle.
F	The robot detected free space and changed lane again to overtake the third dynamic obstacle.
G	Similar to A and D, the robot drove outside of the track before entering the curve to reduce angular acceleration.
H	The robot ramped down the linear velocity and stopped at the destination pose.

Another interesting autonomous driving use case is a robot exploring inside an open-world scenario. For this purpose, the experiment depicted in Figure 14 was used as a validation scenario to test the methodology in open-world scenarios.

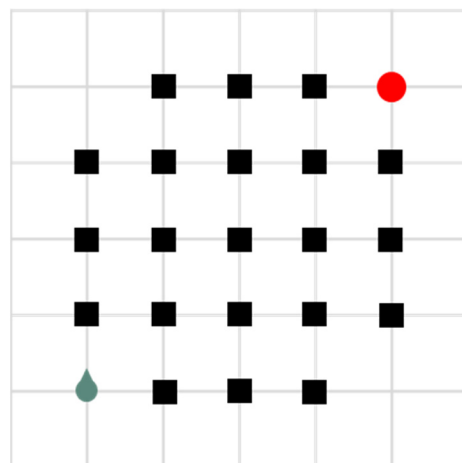


Figure 14. Validation scenario for open-world use cases. The destination pose is marked with a red spot. The robot was intended to avoid all obstacles under safety, comfort, and legal conditions.

The scenario used for the learning process was a simplified open-world scenario to speed up the learning process. The results after the learning process was completed are shown in Figure 15 for different destination poses.

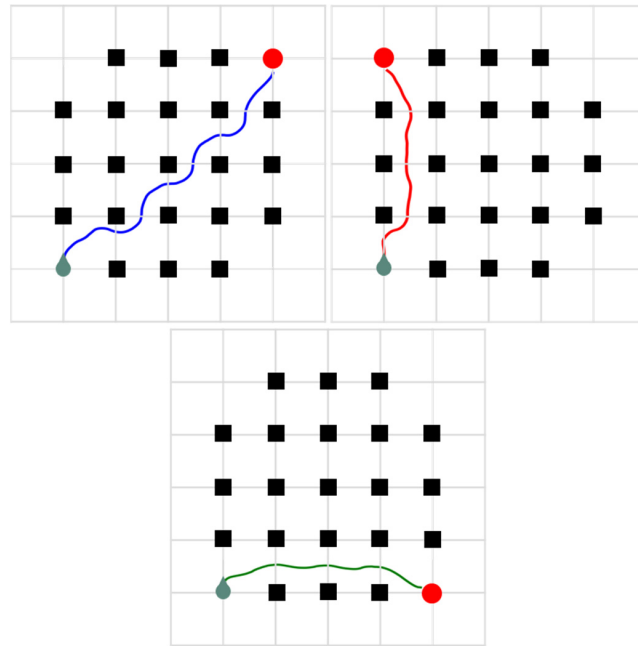


Figure 15. Traveled path (blue, red and green) by the robot in the validation scenario, considering different destination poses. The destination pose is marked with a red spot. The green spot shows the initial position of the robot.

In all three cases from Figure 15, the robot reached the destination pose under safety, legal, and comfort conditions. Nevertheless, the robot still entered at some points (beginning of blue path) the safety margin and got close to some obstacles. This was because the termination conditions of the learning process were met while the agent was still learning. Figure 16 shows precisely that the agent was still learning since the episodic reward was still not flat after more than 200 learning episodes (red line).



Figure 16. Episodic reward versus learning episodes of the learning scenario for the open-world scenario. The red line represents the case where all requirements (safety, legal, comfort, and task-oriented) were implemented. The agent was still learning after the termination condition was met since the slope of the curve was not flat. On the other hand, the blue line, representing the case where only safety and task-oriented requirements were implemented, was flat after the termination condition was met, ensuring that the agent had completed the learning process.

However, since the learning process took a long time, a relax termination condition needed to be defined to find a trade-off between the performance of the resulting policy and the total learning time. In this case, the total learning time went up to 6.1 h. In the case of using of a more powerful system with a server and GPUs, the termination condition could have been adapted to check that the episodic reward does not increase over the learning episodes and ensure in that way a complete learning process.

Finally, a comparison was performed between different strategies to create reward functions. First, a mix of positive (robot driving at desired speed) and negative (collision, comfort, legal, and direction towards the destination) reward functions was created, such as the ones used in many algorithms of the literature reviewed in the section Materials and Methods. Second, a variant with strictly negative reward functions, such as the one proposed in this paper, was chosen as a reference for comparison. The results are shown in Figure 17. The variant with positive and negative terms found a suboptimal solution where the robot keeps driving to collect more rewards.

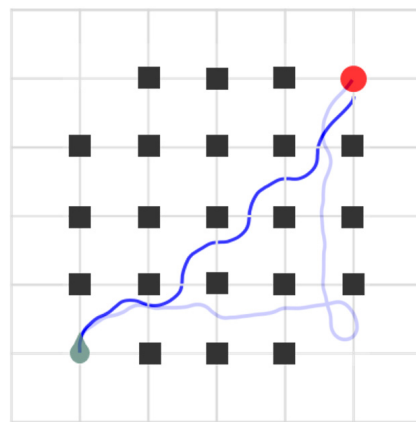


Figure 17. Comparison between different strategies when generating reward functions. Dark blue corresponds to strictly negative reward functions. Light blue corresponds to a mix of positive and negative reward functions.

These results show the effectiveness of the reinforcement learning techniques in dynamic scenarios, which are usually the most complex to solve, and open-world scenarios, which are common for robots whose task is to explore the surroundings. Maneuvers such as obstacle avoidance, overtaking, follow up, lane keeping, or lane change were successfully validated in the previous scenarios.

4. Discussion

The results of the previous section were compared with the dynamic window approach (DWA), which is a state-of-the-art algorithm that belongs to the reactive paradigm group. The DWA algorithm constructs a set of trajectories based on the velocity space. Only velocities that produce safe trajectories are used (i.e., velocities that allow the robot to stop without collision in case of facing a static obstacle). Then, the best trajectory is selected based on the score produced by a well-designed cost function. The results of the DWA algorithm in the dynamic environment are shown in Figure 18.

If the robot finds enough free space, the DWA algorithm succeeds in finding the path to the destination, as shown in mark C. The robot found here enough free space to get through the obstacles, despite being surrounded by dynamic obstacles. On the other hand, if the robot gets trapped between static and dynamic obstacles, the robot fails to find a feasible path and collides with obstacles. Mark A shows when the robot collided with the dynamic obstacle. As a consequence of that, the robot moved backwards to mark B. From mark B onwards, the robot moved forward again to reach the destination. The comparison shows how the reinforcement learning algorithm found a way to adapt the robot's linear velocity to follow up the dynamic obstacle until free space was detected again. This way, a

collision was avoided and the total time to destination was reduced, outperforming the results obtained with the DWA algorithm.

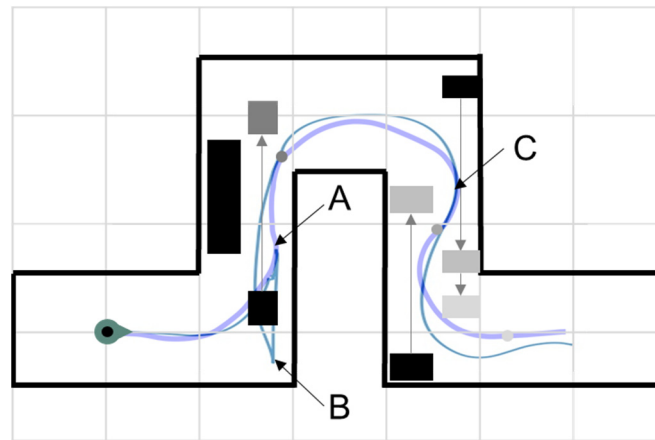


Figure 18. Traveled path by the robot for the reinforcement learning algorithm (light blue) and DWA algorithm (dark blue). Mark A shows a collision with the dynamic obstacle in the DWA algorithm, being forced to move the robot backwards to mark B. Mark C shows a successful overtake of the dynamic obstacles.

Scenarios with dynamic obstacles are very suitable for use cases involving passenger vehicles under real traffic conditions. Deep reinforcement learning models can learn the non-holonomic constraints of a front steering robot, even under conditions close to the grip limit, and solve complex driving scenarios, such as obstacle avoidance with dynamic obstacles, lane keeping, lane change, or overtaking maneuvers.

5. Conclusions

If we think of autonomous driving, many requirements need to be considered, making the development of software a challenging task. Not only do safety requirements need to be considered (lane keeping or avoiding static and dynamic obstacles) but comfort, legal (maximum speed limit), and task-oriented requirements are also essential to be fulfilled. The state-of-the-art solutions produce thousands of possible trajectories to finally select the best one based on the score of a well-designed cost function. However, there are many disadvantages with these state-of-the-art solutions. They are not only very computationally demanding but also very challenging to implement to fulfill all predefined requirements. The self-learning navigation algorithm introduced in this work comes up automatically with a policy that fulfills all predefined requirements at a reasonable computational cost. These requirements are defined by multiple, intuitive, and simple reward functions. Therefore, a highly mature autonomous driving logic or policy is obtained just by trial and error. This methodology has been proven by setting up some classical dynamic scenarios for the autonomous driving domain with maneuvers such as overtaking, following up, lane keeping, or lane changing. Within this section, Figures 12 and 13 show how the safety, legal, and comfort requirements were fulfilled along the entire trajectory driven by the robot. Additionally, the proposed methodology was compared with the dynamic window approach (DWA) algorithm (Figure 18), which is a classic state-of-the-art algorithm to drive robots autonomously while avoiding obstacles. The comparison shows how the reinforcement learning algorithm outperformed the DWA algorithm. Figure 15 also proves that the methodology works effectively in open-world scenarios, avoiding obstacles and driving up to the destination pose under safety, legal, and comfort conditions. The results also show that the learning process can be very time-consuming. However, it can be automated by simulation, loading multiple and complex dynamic driving scenarios. As the autonomous driving logic is self-learned, engineering time and costs can be saved through the software development process. Additionally, the reinforcement learning algorithm

produces a policy considering the dynamic constraints of the non-holonomic robot. This is because the agent interacts and learns directly from the environment where the robot belongs, making it unnecessary to implement embedded models of robot dynamics like in the predictive control model approach.

Author Contributions: Conceptualization, B.P.G.d.l.H., R.M.-T. and J.M.C.T.; validation, B.P.G.d.l.H., R.M.-T. and J.M.C.T.; formal analysis, B.P.G.d.l.H., R.M.-T. and J.M.C.T.; writing—review and editing, B.P.G.d.l.H., R.M.-T. and J.M.C.T.; supervision, R.M.-T. and J.M.C.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are openly available in <https://github.com/bpintos/smart-trajectory-planning> (accessed on 20 December 2023).

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Warren, C.W. Multiple robot path coordination using artificial potential fields. In Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, OH, USA, 13–18 May 1990.
- Fiorini, P.; Shiller, Z. Motion Planning in Dynamic Environments Using Velocity Obstacles. *Int. J. Robot. Res.* **1998**, *17*, 760–772. [[CrossRef](#)]
- Fox, D.; Burgard, W.; Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robot. Autom. Mag.* **1997**, *4*, 23–33. [[CrossRef](#)]
- Álvarez-Sánchez, J.; De la Paz López, F.; Troncoso, J.; Sierra, D. Reactive navigation in real environments using partial center of area method. *Robot. Auton. Syst.* **2010**, *58*, 1231–1237. [[CrossRef](#)]
- Tobaruela, J.A.; Rodríguez, A.O. Rodríguez Reactive navigation in extremely dense and highly intricate environments. *PLoS ONE* **2017**, *12*, e0189008.
- Dreyfus, S.E. An Appraisal of Some Shortest Path Algorithm. *Oper. Res.* **1969**, *17*, 395–412. [[CrossRef](#)]
- Moses, B.; Jain, L.; Finn, R.; Drake, S. Multiple UAVs path planning algorithms: A comparative study. *Fuzzy Optim. Decis. Mak.* **2008**, *7*, 257–267.
- Dolgov, D.; Thrun, S.; Montemerlo, M.; Diebel, J. Practical Search Techniques in Path Planning for Autonomous Driving. In *AAAI Workshop—Technical Report*. 2008; Available online: <https://api.semanticscholar.org/CorpusID:16143233> (accessed on 29 December 2023).
- LaValle, S.M.; Kuffner, J.J. Randomized Kinodynamic Planning. *Int. J. Robot. Res.* **1999**, *20*, 378–400. [[CrossRef](#)]
- Kavraki, L.E.; Svestka, P.; Latombe, J.-C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* **1996**, *12*, 566–580. [[CrossRef](#)]
- Bautista, D.G.; Pérez, J.; Milanes, V.; Nashashibi, F. A Review of Motion Planning Techniques for Automated Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2015**, *17*, 1135–1145.
- Pepy, R.; Lambert, A.; Mounier, H. Path Planning using a Dynamic Vehicle Model. In Proceedings of the 2006 2nd International Conference on Information & Communication Technologies, Damascus, Syria, 24–28 April 2006.
- Paden, B.; Čáp, M.; Yong, S.Z.; Yershov, D.; Frazzoli, E. A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. *IEEE Trans. Intell. Veh.* **2016**, *1*, 33–55. [[CrossRef](#)]
- Pereira, G.C. Lateral Model Predictive Control for Autonomous Heavy-Duty Vehicles: Sensor, Actuator, and Reference Uncertainties. Ph.D. Thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2020.
- González, D.S.; Pérez, J.; Lattarulo, R.; Montero, V.M.; Nashashibi, F. Continuous curvature planning with obstacle avoidance capabilities in urban scenarios. In Proceedings of the 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Qingdao, China, 8–11 October 2014; pp. 1430–1435.
- Farouki, R.T.; Sakkalis, T. Pythagorean hodograph space curves. *Adv. Comput. Math.* **1994**, *2*, 41–66. [[CrossRef](#)]
- Xu, W.; Wei, J.; Dolan, J.; Zhao, H.; Zha, H. A Real-Time Motion Planner with Trajectory Optimization for Autonomous Vehicles. In Proceedings of the 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 14–18 May 2012.
- Svensson, L.; Bujarbaruah, M.; Karsolia, A.; Berger, C.; Törngren, M. Traction Adaptive Motion Planning at the Limits of Handling. *IEEE Trans. Control Syst. Technol.* **2020**, *30*, 1888–1904. [[CrossRef](#)]
- Zhou, C.; Huang, B.; Fränti, P. A review of motion planning algorithms for intelligent robots. *J. Intell. Manuf.* **2022**, *33*, 387–424. [[CrossRef](#)]
- Nair, R.S.; Supriya, P. Robotic Path Planning Using Recurrent Neural Networks. In Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020.

21. Hu, Y.; Yang, L.; Lou, Y. Path Planning with Q-Learning. *J. Phys. Conf. Ser.* **2021**, *1948*, 012038. [[CrossRef](#)]
22. Lillicrap, T.; Hunt, J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
23. Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.-M.; Lam, V.-D.; Bewley, A.; Shah, A. Learning to Drive in a Day. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.
24. Vitelli, M.; Nayebi, A. CARMA: A Deep Reinforcement Learning Approach to Autonomous Driving. 2016. Available online: https://anayebi.github.io/files/projects/CS_239_Final_Project_Writeup.pdf (accessed on 29 December 2023).
25. Wang, P.; Chan, C.-Y.; de La Fortelle, A. A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018.
26. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the 1st Annual Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017; pp. 1–16.
27. Kardell, S.; Kuosku, M. Autonomous Vehicle Control via Deep Reinforcement Learning. Master’s Thesis, Chalmers University of Technology, Gothenburg, Sweden, 2017.
28. Chen, J.; Yuan, B.; Tomizuka, M. Model-free deep reinforcement learning for urban autonomous driving. In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 2765–2771.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.