




Article

# Stability Analysis for Autonomous Vehicle Navigation Trained over Deep Deterministic Policy Gradient

Mireya Cabezas-Olivenza <sup>1</sup>, Ekaitz Zulueta <sup>1,\*</sup>, Ander Sanchez-Chica <sup>1</sup>, Unai Fernandez-Gamiz <sup>2</sup>  
and Adrian Teso-Fz-Betoño <sup>1</sup>

<sup>1</sup> System Engineering and Automation Control Department, University of the Basque Country (UPV/EHU), Nieves Cano, 12, 01006 Vitoria-Gasteiz, Spain

<sup>2</sup> Department of Nuclear and Fluid Mechanics, University of the Basque Country (UPV/EHU), Nieves Cano, 12, 01006 Vitoria-Gasteiz, Spain

\* Correspondence: ekaitz.zulueta@ehu.eus

**Abstract:** The Deep Deterministic Policy Gradient (DDPG) algorithm is a reinforcement learning algorithm that combines Q-learning with a policy. Nevertheless, this algorithm generates failures that are not well understood. Rather than looking for those errors, this study presents a way to evaluate the suitability of the results obtained. Using the purpose of autonomous vehicle navigation, the DDPG algorithm is applied, obtaining an agent capable of generating trajectories. This agent is evaluated in terms of stability through the Lyapunov function, verifying if the proposed navigation objectives are achieved. The reward function of the DDPG is used because it is unknown if the neural networks of the actor and the critic are correctly trained. Two agents are obtained, and a comparison is performed between them in terms of stability, demonstrating that the Lyapunov function can be used as an evaluation method for agents obtained by the DDPG algorithm. Verifying the stability at a fixed future horizon, it is possible to determine whether the obtained agent is valid and can be used as a vehicle controller, so a task-satisfaction assessment can be performed. Furthermore, the proposed analysis is an indication of which parts of the navigation area are insufficient in training terms.



**Citation:** Cabezas-Olivenza, M.; Zulueta, E.; Sanchez-Chica, A.; Fernandez-Gamiz, U.; Teso-Fz-Betoño, A. Stability Analysis for Autonomous Vehicle Navigation Trained over Deep Deterministic Policy Gradient. *Mathematics* **2023**, *11*, 132. <https://doi.org/10.3390/math11010132>

Academic Editors: Costin Badica, Nick Bassiliades, Kalliopi Kravari and Theodoros Kosmanis

Received: 22 November 2022

Revised: 22 December 2022

Accepted: 23 December 2022

Published: 27 December 2022



**Copyright:** © 2022 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:** navigation; neural network; autonomous vehicle; reinforcement learning; DDPG; lyapunov; stability; q-learning

**MSC:** 34D23

## 1. Introduction

As autonomous vehicles (AVs) are increasingly used in industrial environments, there is a need to develop intelligent and adaptable navigation systems for these industrial scenarios. To accomplish this, vehicles need to be able to generate the necessary actions to, for example, avoid collisions. This implies that autonomous vehicles need real-time decision-making capabilities. Therefore, it is necessary to take immediate measurements of the environment. These measurements will be provided by sensors embedded in the vehicles themselves and read by a navigation algorithm. The algorithm must also have the vehicle's dynamic characteristics implemented to be able to decide the appropriate control signals at any given point in time. One possible solution for managing these control signals is the use of neural networks.

One of the dynamic properties to be considered in these applications is the type of wheels to be used. For instance, mecanum wheels are commonly unemployed. It should be taken into account that this type of wheel frequently produces arbitrary slippage and vibrations at high speeds. As Xie et al. [1] analysed, this leads to positioning errors and energy consumption problems in complex tasks. In order to improve these aspects, Piemngam et al. [2] show the development of a platform for the navigation of an autonomous vehicle with mecanum wheels, including the mechanical design, the system design, and

the construction of the vehicle. The analysis carried out by Kim et al. [3] proposes the use of INS in vehicles with this type of wheel. It integrates an encoder, an accelerometer, and a gyro sensor through two Kalman filters and can make decisions regarding sliding and positioning corrections.

It is equally possible to implement the A\* algorithm by accounting for the working environment of the mecanum wheeled mobile robot. Li et al. [4] studied this algorithm due to the fact that it has good expansibility and adaptability, achieving navigation in simulation. Moreover, it is feasible to use the Dijkstra algorithm as a global route planner and the DWA for local routes. A map is constructed through SLAM (Simultaneous localization and mapping), being able to bypass obstacles and reach the target (see Liu et al. [5]). Note that the A\* algorithm is a generalization of the Dijkstra algorithm. In the latter, the heuristic function  $h(n)$  is zero, while A\* uses a non-zero, normally admissible heuristic.

Continuing with methodologies for autonomous vehicle navigation, one of the most common is the Dynamic Window Approach (DWA). With this technique, it is necessary to consider the restrictions imposed by velocities and accelerations, with these values being admissible within the dynamic window. In the context of robots with synchronised drive, Fox et al. [6] consider the dynamics of the autonomous vehicle to take into account the inertia and to be able to control at high speeds. Furthermore, as demonstrated by Brock [7], it is possible to combine real-time motion planning and obstacle avoidance methods, with the former being based on the map of the environment and the latter on sensory information. This provides a robust method at high speeds. Saranrittichai et al. [8] present a method called Field Dynamic Window Approach (F-DWA), where the objective function is modified by considering the obstacles near the trajectory. Furthermore, returning to mecanum wheels, Xie et al. [9] present an extension of the DWA by considering energy, and thus realising a combination of objectives in the cost function.

Disregarding vehicle dynamics and aiming at fulfilling navigation in an autonomous vehicle, different techniques can be applied. The use of the Vector Field Histogram (VFH) has proven useful for this purpose, as demonstrated by Borenstein et al. [10]. By mapping the environment through a two-dimensional cartesian histogram grid, they obtain the model of the scenario. Afterwards, two-step data reduction processes are applied, and the desired control commands can be obtained. Those are obtained by posing the value of the obstacle density per sector and aligning the robot direction with a low value of the obstacle density. Ye et al. [11] demonstrate that the Traversability Field Histogram (TFH) is also applicable for navigation. A two-dimensional laser range finder (LRF) is used to obtain field features and is then transformed to extract slope and roughness data. Through the Polar Traversability Index (PTI), which symbolises the difficulty of navigating in a given direction, it is possible to correct the shortcomings of the VFH.

It is also possible to construct a viable means of navigation through the combination of the Virtual Force Field (VFF) and the Potential Flow Field (PFF). The first one provides heading angles when determining collision risks, while the second one provides heading angles derived from a velocity vector (see Burgos et al. [12]). Wang et al. [13] demonstrate that with VFF detailed route planning is possible, taking into account dynamic obstacles. A two-layer designed system is presented in which route planning decisions are made and route tracking is controlled by MPC predictive control. In another study (see Maarif et al. [14]), the artificial potential field (APF) is additionally used as a method to solve real-time navigation, determining that the method needs modifications to overcome local minima problems.

The use of lateral control has also been implemented to comply with navigation. Hoffmann et al. [15] present a non-linear control law, considering the direction of the front wheels instead of the vehicle structure, with respect to the desired trajectory. Using a proportional integral (PI) controller, they achieve speed control by acting on the brake and accelerator. This algorithm is commonly known as the Stanley method. In addition, it is possible to demonstrate that, with the use of computer vision, it is a highly robust algorithm that produces a minimum deviation error value with respect to the trajectory (see

Cabezas-Olivenza et al. [16]). Instead of using computer vision, it is possible to use GPS sensors, IMU, LiDAR, etc. as Abdelmoniem et al. [17] present in the study. Using a local time elastic band (TEB) planner, it is feasible to provide the controller with a collision-free trajectory to follow with the Stanley method.

Alternatively, if a dynamic collision model capable of predicting future collisions is generated, it is possible to complete navigation, with the controller output being non-holonomic accelerations (see Missura et al. [18]). If the MPC/CLF framework is used as a reference, a treatment of the convergence properties of the algorithm can also be performed, as shown by Ögren et al. [19]. Kashyap et al. [20], for humanoid robots, describe a combination between DWA and a teaching-learning-based optimisation (TLBO) technique. In this case, the DWA is responsible for providing the optimal speed parameter, and that result feeds the TLBO to obtain the turning angle. In this way, obstacles are avoided while moving towards the target. With the use of a deep convolutional neural network, it is equally possible to make an optimal choice of DWA parameters, as demonstrated by Dobrevski et al. [21]. The network is trained with a reinforcement learning algorithm, and the parameters are dynamically predicted by considering the sensor readings. In this way, adaptation to the environment is achieved and smoother trajectories can be realised. Reinforcement learning is also applied in the study by Chang et al. [22]. Evaluation functions are added to improve navigation, while the DWA parameters are adaptively learned by Q-Learning.

Continuing with reinforcement learning methods, the Deep Deterministic Policy Gradient (DDPG) technique is presented. This method has been experimented with for navigation in various studies. For instance, together with a continuous action space technique to train the mobile robot to navigate through or over obstacles (see Bouhamed et al. [23]). Rewards are consequently given for distance to the destination and penalties for collisions. These rewards can also be vector-based, as demonstrated by You et al. [24], where a Deep Reinforcement Learning (DRL) -based navigation decision framework is proposed to validate the assignments.

In the study presented by Zhang et al. [25], obstacle avoidance and navigation in a robotic control area are performed. By merging the DDPG with a proximal policy optimization (PPO) and an improved reward model technique, it manages to reach the target point. On the other hand, Xie et al. [26] find there is indeed a shortage of rewards in areas with a significant degree of environmental variation. This means the DDPG has variance problems in complex environments, so a stochastic switch incorporation is proposed. This allows the agent to select between high and low-variance policies. To perform this, the stochastic switch is trained at the same time as the DDPG, and the robot dynamically chooses which method to learn.

Gao et al. [27] propose a DDPG framework with experience separation because, conventionally, the DDPG stores the exploration experience in a buffer without considering whether the transition is valuable. With such separation, it can discretely reproduce valuable and failed experience transitions. On the other hand, the use of wireless sensor networks (WSNs) requires real-time data transfers. Therefore, Bouhamed et al. [28] propose an autonomous mobile robot data collection mechanism for delay-tolerant WSN applications. To this end, DDPG and Q-Learning are employed simultaneously in the training of the robot, with the first one deciding the best trajectory and the second one determining the order of the nodes to be moved.

Finally, if one wants to translate the DDPG technique to the real world, one has to take into account the amount of data in the environment, as the agent learns the optimal action strategy in simulation environments (see Guo et al. [29]). The DDPG can be enhanced for this purpose by coupling it with an artificial potential field, with a path planning model that is integrated into the platform. Liu et al. [30] design a method called Reinforcement Driving that serves for the transmission of well-trained models to the real world. First, an agent is trained in simulation, which will then evaluate the real scenario.

From the bibliography, as it can be appreciated, there is a tendency to merge various techniques with the DDPG algorithm to carry out the navigation of autonomous vehicles. However, there is a perceived deficiency in verifying the quality of the agent for each application. It is unproven whether the agent obtained from the training of the DDPG, which is the one that is going to direct the control of the vehicle, is suitable or not.

However, studies have focused on the analysis of the failures that occur in the DDPG algorithm, as the reason for these failures is poorly understood. Matheron et al. [31] have attempted to explain the faults in the case of sparse rewards and deterministic environments. In this way, one of the convergence regimes of this algorithm has been understood. Furthermore, Grando et al. [32] demonstrate that the tendency to use simple sensing strategies leads to poor performance in high-dimensional state spaces. They compare the DDPG with the Soft Actor-Critic (SAC) technique, demonstrating the influence of the neural networks used in the executions.

It is recognised that experiential learning techniques often fail to learn an effective policy, so a technique known as imitation learning can be applied. Unlike DDPG, rather than receiving rewards, it is based on a learn-and-reproduce method, as explained by Kormushev et al. [33], who, in their study, encode it as a mixture of dynamic systems. Imitation can be an efficient method for agents to learn by giving them a set of demonstrations of what they have to acquire. It is feasible to refine the supervision policy by using active learning (see Hussein et al. [34]). It is then generalised so that it can be used in unknown environments. This technique uses deep convolutional neural networks.

Additionally, Inverse Reinforcement Learning has also been a subject of study. The IRL is assumed to behave according to an intrinsic cost function that reflects its intent and informs its control actions. Tesfazgi et al. [35] analyse how to infer the intention of an intelligent agent from demonstrations and subsequently predict its behaviour. To do so, they reformulate the IRL inference problem to learn the Lyapunov function as a control, thus ensuring stability under inference control policies. In the Choi et al. [36] study, it is applied to navigation and aims to learn the control performance of an expert through demonstrations, that is, imitations. For this purpose, the trajectory is obtained with a Hidden Markov Model (hmm) and Dynamic Time Warping (DTW). To design the controller, a hidden reward function of a quadratic form is learned. Furthermore, the reward function that minimises the trajectory tracking error is chosen.

Nevertheless, these are different techniques from the DDPG applied in this study. In contrast to imitation learning, in this case, the desired trajectory is not known but will be generated through rewards. About Inverse Reinforcement Learning, there is no known policy for this application, so it is not possible to obtain the best reward function.

The present work utilizes the dynamic model of the autonomous vehicle presented in the BCCPS&AI conference proceedings (see Graña [37]), where the dynamics of the DWA algorithm are designed for vehicles with mecanum wheels. With this dynamic, the DDPG algorithm is implemented to obtain an agent that proposes the navigation trajectory, training the corresponding neural networks. Once the trajectory is acquired, a stability analysis of the system is performed in a bounded environment. Using Lyapunov's method, it is possible to demonstrate the system obtained is stable. For this purpose, the reward function defined for the DDPG training is applied. Therefore, the aim is to evaluate the quality of the agent through the stability of the trajectories it proposes in a given horizon, which ensures that the vehicle eventually reaches its destination.

## 2. Materials and Methods

Autonomous vehicles or AVs require a control algorithm to perform the optimal trajectory to reach a target point. This algorithm requires considering the structure and equipment of the vehicle, which in this article is about the mecanum wheels.

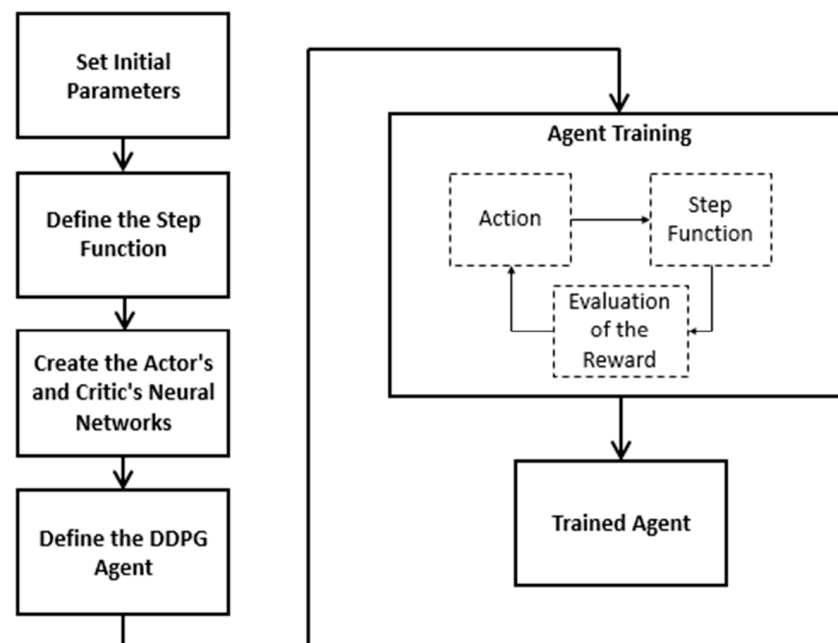
To achieve collision-free navigation, it is possible to manage neural networks. In this case, the algorithm applied is the DDPG (see Lillicrap [38]), which is trained to provide agents that generate optimal trajectories. After getting the trajectory, it is necessary to

validate if the design obtained is stable, so the Lyapunov method is employed. A reward function is employed to train the network, which will posteriorly be analysed with the Lyapunov energy function. In this manner, the quality of each agent obtained is evaluated.

Lyapunov stability theory was developed by Lyapunov in 1892. By means of it, it is possible to determine the stability of non-linear systems through the Lyapunov energy functions. It is therefore known to be a tool to help determine whether a system is stable or not (see [39–41]). In this study, an experimental method to verify stability is proposed. As mentioned, the trajectories are obtained through DDPG agents, which remain dynamic systems. In this way, the Lyapunov method is applicable.

Therefore, Lyapunov's theory allows ensuring that, if the reward function satisfies Lyapunov's conditions of temporal decrement, the convergence of this function can be guaranteed by reinforcement learning. It should be mentioned that the reward function represents a weighted average of a series of rewards over a time horizon. If this reward function is treated as a Lyapunov function, it can be ensured that the agent will always improve at a certain horizon. The Lyapunov method is intended to be used as a comparison tool between proposed solutions.

The objective is to develop a tool to be used for evaluating the agents obtained through the DDPG training process. First, it is necessary to obtain these agents through the Reinforcement Learning training process. A flow chart is presented in Figure 1, which clarifies this process. Initially, it is required to establish the initial parameters, such as the definition of the navigation area and the initial conditions of the navigation for the autonomous vehicle. This is preceded by the definition of a function that will be referred to as a Step Function. This function contemplates both the dynamics of the vehicle and the reward function that evaluates how well the movement has been. In turn, this function includes termination conditions, by which it is considered that an agent is not performing a good movement. It will also be necessary to create the neural networks corresponding to the actor and the critic of the DDPG. In addition, it will be necessary to define the characteristics of the agent.



**Figure 1.** Flowchart of the DDPG method of agent generation.

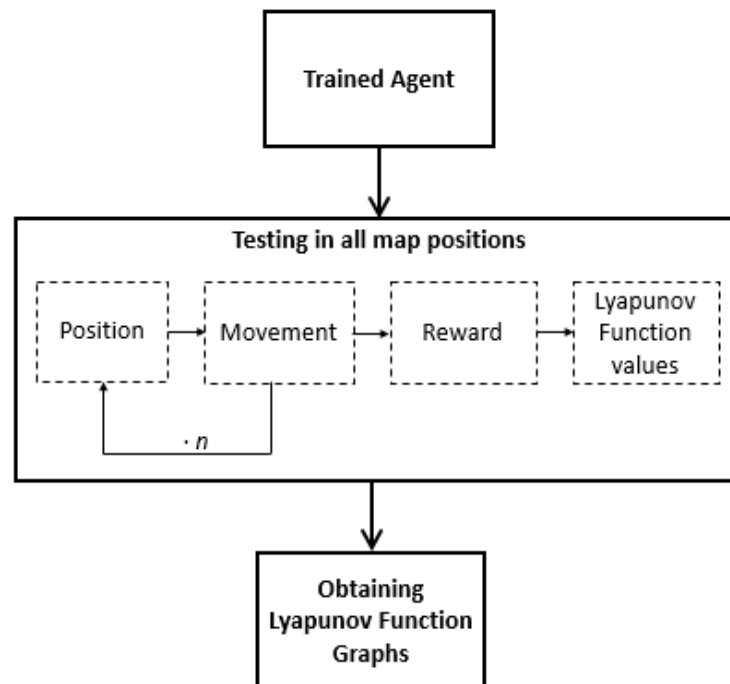
Having completed all the necessary definitions, the DDPG is trained. An action is proposed by the actor. This action is implemented in the Step Function, that is, in the vehicle dynamics. Subsequently, the reward evaluates the movement generated by this action. This reward is judged by the critic, which will indicate to the actor if the proposed action



is good or if it must be varied considerably. This loop is performed for several iterations, as long as the agent does not encounter any of the defined termination conditions. At the end of the training, an agent with an experienced buffer is obtained, which can generate commands for the movement of the mobile robot, that is, a trajectory.

Once the trained agent is obtained, it is evaluated using the Lyapunov method. It is necessary to mention that the objective of navigation is to reach a target position in addition to avoiding objects. In order to check that objective accomplishment, the Lyapunov function is focused on a future horizon of  $n$  steps of the mobile robot or, in other words, on its increment in one known horizon.

In order to clarify the process, Figure 2, which is a flowchart of the procedure of testing the reward function through Lyapunov, is presented. Accordingly, the agent obtained from the previous process, the one defined in the flowchart in Figure 1, is used. This agent is examined in all the positions of a delimited area. It starts from a known position in which the reward value is also available. By means of the dynamics of the vehicle and the experience of the agent,  $n$  steps are made to reach the desired horizon. At this horizon, the reward is re-evaluated, and a comparison is made with the initial position reward, thus obtaining values for the Lyapunov function. Once the Lyapunov values are achieved for the entire defined area, the surface graphics are generated.



**Figure 2.** Flowchart of the Lyapunov function surface graphics obtained.

Through this process, it is possible to evaluate the effectiveness of an agent with the system it can generate, verifying its stability.

Regarding the coding of both flowcharts, the pseudocodes are presented below. For the flowchart in Figure 1, the pseudocode in Figure 3 is offered. It shows the base programming that must be followed to obtain an agent with an experienced buffer through the DDPG deep learning process. As can be observed, there is no loop, as the whole process is successive. The only repetitive operation is the agent’s training, which accesses the Step function numerous times to evaluate actions.

In reference to the flowchart in Figure 2, the pseudocode in Figure 4 is presented; once the agent has been obtained, it is evaluated by means of the Lyapunov function. In this pseudocode, the agent is processed, and, through this variable, it is possible to obtain the trained networks of the critic and the actor. In this way, a loop is generated so that the Lyapunov function is checked in all the positions of a defined area. Once the reward value

of a position has been calculated, the AV is advanced to a future horizon, through a loop, to determine the reward value at a future position. Finally, the graphs that can be seen in Section 3 are produced.

---

**Agent generation through the DDPG algorithm**

---

- 1: Definition of global variables (map and obstacles, max speed, starting random position, target position)
- 2: Define number of observations
- 3: Define number of actions and their limits
- 4: Reset variables
- 5: Create the environment with Step Function
- 6:       Obtain the current status of the AV
- 7:       Apply vehicle dynamics
- 8:       Update AV status
- 9:       Check break conditions
- 10:       Calculate the *reward*
- 11: Create the critic's neural network *criticNetwork* and define the critic
- 12: Create the actor's neural network *actorNetwork* and define the actor
- 13: Create the agent
- 14: Train the agent through the environment
- 15: Simulate agent results

---

Figure 3. Agent procurement process pseudocode.

---

**Application of the Lyapunov function**

---

- 1: Definition of global variables (map and obstacles, max speed, target position, time step)
- 2: Load the agent
- 3: Load actor's network from agent variable
- 4: Load critic's network from agent variable
- 5: Create vector with all map positions
- 6: for  $i \leftarrow$  all map positions
- 7:       Get *reward* value (*position i*) with *criticNetwork*
- 8:       Store *reward* value as Lyapunov function value
- 9:       *position j = position i*
- 10:       for  $j = 1$  to  $n$
- 11:       |       Get action (*position j*) with *actorNetwork*
- 12:       |       Calculate *nextPosition* through the vehicle dynamics
- 13:       |       *position j = nextPosition*
- 14:       end
- 15:       Get *reward* value (*nextPosition*) with *criticNetwork*
- 16:       Calculate the *nextPosition reward*
- 17:       Store Lyapunov increment as (*nextPosition reward*)-(*reward*)
- 18: end
- 19: Obtain graphs of Lyapunov function and its increment

---

Figure 4. Pseudocode of Lyapunov function computation on the agent.

With two individual files programmed in this mode, it is therefore possible to perform the evaluation of the agents obtained through the DDPG.

### 2.1. Vehicle Dynamic Model

The mecanum wheels are designed to allow the autonomous vehicle to perform translational movements in any direction in combination with turning on itself, as shown in Figure 5. Consequently, it can be considered that it will have two linear velocities,  $V_x$  [m/s] and  $V_y$  [m/s]. Moreover, the vector formed by both will be the linear velocity  $V$  [m/s], which determines the direction adopted by the autonomous vehicle. This direction will clearly be influenced by the direction adopted by the autonomous vehicle, that is, the value of  $\vartheta_v$  [rad].

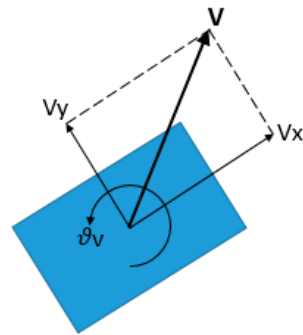


Figure 5. Different speeds involved in the movement of the autonomous vehicle.

Considering the values in Figure 5, it is possible to define a vector that collects them, as shown in Equation (1). The action vector  $U$  can equally be applied to all the states of the trajectory, denoted as  $t_k$ .

$$U(t_k) = \begin{bmatrix} V_x(t_k) \\ V_y(t_k) \\ \vartheta_v(t_k) \end{bmatrix} \tag{1}$$

In this way, the three parameters that will act on the movement of the vehicle are defined.

#### Vehicle Control Model

Knowing the parameters which act on the mobile robot, it is necessary to formulate their dynamics in order to be able to identify the positions of the mobile robot in the future states. Equations (2)–(5) are therefore formulated, where  $x$  [m] and  $y$  [m] refer to vehicle positions on the plane, while  $\Delta t$  [s] is the integration step.

$$x(t_k + 1) = x(t_k) + U_1 \Delta t \tag{2}$$

$$y(t_k + 1) = y(t_k) + U_2 \Delta t \tag{3}$$

Equations (2) and (3) may depend on the value of  $U_3$  as shown in Equations (4) and (5).

$$x(t_k + 1) = x(t_k) + V(t_k) \cos(U_3) \Delta t \tag{4}$$

$$y(t_k + 1) = y(t_k) + V(t_k) \sin(U_3) \Delta t \tag{5}$$

To calculate the value of  $\vartheta$  [rad], which is the vehicle angle on the plane, a different approach is used at each instant. To begin with, the value of the relative position of the autonomous vehicle is considered. This location, noted as  $RelPos$ , remains the Euclidean norm of differences between the target position and the actual position of the vehicle, defined as in Equation (6).

$$RelPos(t_k) = \|(x_{target}, y_{target}) - (x(t_k), y(t_k))\| \tag{6}$$



With this value, a new parameter called  $fd$  [m] is created, which will help to obtain the value of  $\vartheta$  [rad]. For this purpose, a constant called  $\alpha_3$  is used, and Equation (7) is stated. The value of the constant is  $\alpha_3 = 0.1$ .

$$fd = 1 / (1 + \exp(-\alpha_3 RelPos(t_k))) \tag{7}$$

The need for a collision-free trajectory is accounted for. Therefore, the following Equation (8) is proposed. Therein, the values of  $V_x$  [m/s] and  $V_y$  [m/s] are obtained as shown in Equations (9) and (10). The value  $\vartheta_{target}$  [rad] refers to the desired angle of the autonomous vehicle's arrival at the destination.

$$\vartheta(t_k + 1) = fd(\text{atan}\left(\frac{V_y}{V_x}\right)) + (1 - fd)\vartheta_{target} \tag{8}$$

$$V_x(t_k) = V(t_k) \cos(U_3)\Delta t \tag{9}$$

$$V_y(t_k) = V(t_k) \sin(U_3)\Delta t \tag{10}$$

Therefore, the control law by which the mobile robot will be governed is defined. It can be seen how everything depends on  $U_3$ , which will be the action to be taken through the DDPG.

### 2.2. DDPG Training

Once the control law is established, it is possible to perform reinforcement learning based thereon. The method utilized in this article is the Deep Deterministic Policy Gradient (DDPG). For that reason, it is necessary to define a reward function so that the agent learns the action to be taken in the following instance. The agent, in this case, represents the autonomous vehicle with mecanum wheels, which will move according to the dynamics designed in the previous section.

The environment defined for navigation is depicted in Figure 6. In this illustration, the map boundary and the obstacles in the environment are shown with blue crosses. The red cross refers to the point from which the vehicle will start navigating, being a sample location in this figure. The green circle is established as a reference for the destination position, although the main objective is also to leave the delimited area without collision. For the development, the obstacles and area boundaries are considered equal, referring to everything as obstacles, hence there is no differentiation in the image.

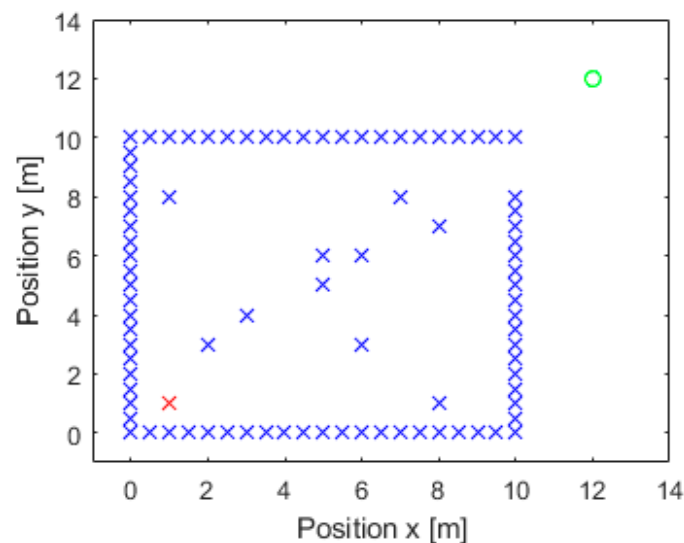


Figure 6. Proposed navigation map, including obstacles, start and end point of navigation, as well as delimited area.

After having established the position of the obstacles, as depicted in Figure 6, different equations can be proposed. The training will be reinforced by the distance to the obstacles and the distance to the destination.

Consequently, it is necessary to calculate the distance between the nearest obstacle to the vehicle and the following position that the AV will adopt. The reason for this is that it is required to evaluate how good that movement is, assessing it with the parameter  $dist_{obs}$  [m]. To do so, it is necessary to know the position of the *obs.* obstacles at  $x$  [m] and  $y$  [m] and the position  $x(t_k + 1)$  [m] and  $y(t_k + 1)$  [m] of the vehicle. Equation (11) is proposed for this purpose. This equation will be applied to all obstacles, keeping only the distance value in reference to the nearest one, noted as  $j$ .

$$dist_{obs} = \min \left\| \left( x_{obs,j}, y_{obs,j} \right) - \left( x(t_k + 1), y(t_k + 1) \right) \right\|, j \in \{1, 2, 3, \dots, N_{obs}\} \quad (11)$$

On the other hand, the same applies, but with the distance between the target and the position after the mobile robot movement. Similarly, it will be necessary to evaluate whether the vehicle is approaching the target with the variable  $dist_{target}$ . Equation (12) proposes how to obtain this value. It will be necessary then to know the position data to be reached, defined as the *target.* in  $x$  [m] and  $y$  [m].

$$dist_{target} = \left\| \left( x_{target}, y_{target} \right) - \left( x(t_k + 1), y(t_k + 1) \right) \right\| \quad (12)$$

In this manner, with knowledge of the map and the characteristics of the movement of the autonomous vehicle, it is possible to define the reward function, noted as *reward*. This will be unitless and, in this case, is set in Equation (13). Each part of the equation is multiplied by a constant depending on the importance of each objective. Those constants are designated as  $\lambda_1$  and  $\lambda_2$ . These remain unitless constants, and, in this case, the decision has been taken to assign them a value of  $\lambda_1 = 5$  and  $\lambda_2 = 0.15$ , since the distance to the target is to be considered more important.

$$reward = \lambda_1 \left( \frac{1}{dist_{target}} \right)^2 + \lambda_2 (dist_{obs})^2 \quad (13)$$

Hence, if the value of the reward increases between  $t_k$  and  $t_k + 1$ , the action is appropriate, while if the value decreases, the action will not be so favourable. Once the reward function has been determined, the networks to be used are specified.

The action proposed by the network depends exclusively on the position of the autonomous vehicle at each moment, so there will be three observation parameters, according to  $U$ . In addition, everything has been defined as dependent on the direction of the vehicle. Therefore, it is decided that there will be only one action, that is, it will act on the vehicle's steering angle  $\theta v$  [rad]. Depending on whether the following position of the vehicle, based on the direction given by the action, is satisfactory, more or less reward will be obtained. Furthermore, it is established that this action can only vary in the range  $[-\pi, \pi]$  radians. This is proposed in accordance with the vehicle's dynamics.

Based on these conditions, the critic's network and the actor's network are shown in Figure 7. First, in Figure 7a, the left branch corresponds to the observation parameters, while the right branch corresponds to the action. Both branches are then connected through an additional layer. Layers FC1, FC2, FC3, FC4, and FC5 correspond to fully connected layers. All FC layers have 10 outputs, except FC4 which has only 1. Relu1, Relu2, and Relu3 correspond to layers with a ReLU activation function. Regarding Figure 7b, the input is the observation. The layers ActorFC1 and ActorFC2 are fully connected layers with 50 outputs, and the layer ActorFC3 corresponds to a fully connected layer with 1 output. ActorRelu1 and ActorRelu2 are layers with a ReLU activation function. Regarding the output of this network, it has an ActorTanh layer, which is a hyperbolic tangent, and a last layer called ActorScaling to make the output scaled to a maximum value of  $\pi$ . Both networks have a Learn Rate of 1, a Gradient Threshold of 1 and an L2 Regularization Factor of 0.01. Table 1

is presented to provide a general approach to the specifications that have been discussed about the neural network of the critic.

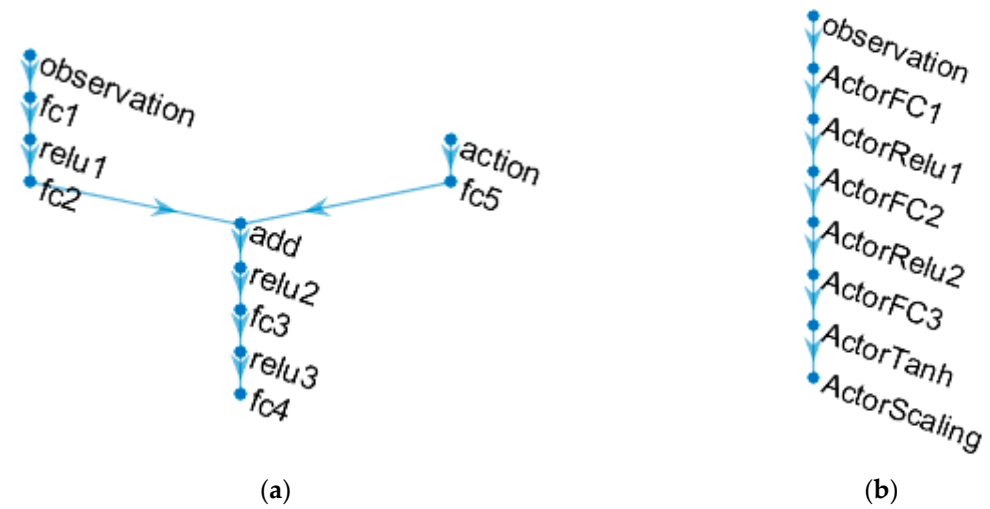


Figure 7. (a) Proposed Critic neural network; (b) Proposed Actor neural network.

Table 1. Characteristics of the critic neural network.

Layer Names	Type	Number of Layers	Learnable Properties
observation	Feature Input	4	-
fc1	Fully Connected	10	Weights $10 \times 4$ Bias $10 \times 1$
relu1	ReLU	10	-
fc2	Fully Connected	10	Weights $10 \times 10$ Bias $10 \times 1$
action	Feature Input	1	-
fc5	Fully Connected	10	Weights $10 \times 1$ Bias $10 \times 1$
add	Addition	10	-
relu2	ReLU	10	-
fc3	Fully Connected	10	Weights $10 \times 10$ Bias $10 \times 1$
relu3	ReLU	10	-
fc4	Fully Connected	1	Weights $1 \times 10$ Bias $1 \times 1$

Table 2 is also presented, as in Table 1, in relation to encompassing the characteristics of the actor’s neural network.

Table 2. Characteristics of the actor neural network.

Layer Names	Type	Number of Layers	Learnable Properties
observation	Feature Input	4	-
ActorFC1	Fully Connected	50	Weights $50 \times 4$ Bias $50 \times 1$
ActorRelu1	ReLU	50	-
ActorFC2	Fully Connected	50	Weights $50 \times 50$ Bias $50 \times 1$
ActorRelu2	ReLU	50	-
ActorFC3	Fully Connected	1	Weights $1 \times 50$ Bias $1 \times 1$
ActorTanh	Tanh	1	-
ActorScaling	ScalingLayer	1	-

As previously mentioned, Figure 7 presents a graphical representation of both neural networks.

Once the networks are defined, it is possible to create the DDPG agent. It will be defined with a Sample Time of 0.1 s. The Target Smooth Factor will have a value of 0.001, while the Discount Factor will be 0.995. A Mini Batch Size of 128 and an Experience Buffer with a length of  $1e^6$  are also defined. Noise with a Variance of 0.1 and a Variance Decay Rate of  $1e^{-5}$  are also included. These data are presented in Table 3.

**Table 3.** DDPG Agent definition parameters.

Parameter Name	Value
Sample Time	0.1
Target Smooth Factor	$1 \times 10^{-3}$
Discount Factor	0.995
Mini Batch Size	128
Experience Buffer Length	$1 \times 10^6$
Noise Variance	0.1
Noise Variance Decay Rate	$1 \times 10^{-5}$

When all the elements are established, the training is executed. It should be noted that the training is performed without altering the value of the objective position, but the values from which the vehicle starts to move, that is, the starting point and the initial angle, are set as random (The randomisation is defined in Section 3). It is also necessary to clarify that the speed set for the autonomous vehicle is constant, since the purpose is not to stop at the destination point but to orient the vehicle towards the outside area. It has been proposed to set this value at  $V = 10$  [m/s].

Additionally, it is necessary to point out that the agent's training has termination conditions, such as leaving the limits of an enclosed area defined between the values of 0 and 14 [m] for both the  $x$  and  $y$  axes. The other stop condition of the training is defined so that when it approaches a distance of less than 0.5 [m] from an obstacle, it terminates.

### 2.3. Stability Analysis

A study is carried out to demonstrate that the trained agent creates stable trajectories. For this purpose, the Lyapunov function is used. In this case, the Lyapunov energy function, parameterized as  $L$ , which will be unitless, is proposed in consideration of the reward function expressed in Equation (13).

First, the vector of states is defined as in Equation (14).

$$s(t_k) = \begin{bmatrix} x(t_k) \\ y(t_k) \\ \vartheta(t_k) \end{bmatrix} \quad (14)$$

In order to clarify how the analysis of the Lyapunov function has been carried out, its application in the DDPG algorithm is explained at a theoretical level. First, Equation (15) is presented. Once the agent has been obtained through the DDPG algorithm, the next state of the vehicle,  $s$ , is obtained through a function that considers the action proposed by the agent, denoted as  $a$ , and the current state in which the AV is.

$$s(t_k + 1) = f(s(t_k), a(t_k)) \quad (15)$$

The agent obtains the action value through the defined actor neural network, denoted as  $\pi$ , which considers the state of the vehicle, as shown in Equation (16).

$$a(t_k) = \pi(s(t_k)) \quad (16)$$

Moreover, it is possible to obtain the value of the reward for all positions in the area. This is performed through the neural network of the critical  $Q$ , which is a network that considers both the action and the current state  $s$  of the vehicle. The result represents the mathematical expectation of the reinforcement over time, with a discount factor of  $\gamma$ . This is defined in Equation (17).

$$E\left(\sum_{i=0}^{i=+\infty} \gamma^i \text{reward}(t_k + i)\right) = Q(s(t_k), a(t_k)) \tag{17}$$

The definition of the DDPG algorithm is determined using these three previous equations. It should be noted that the critical is a trained network that cannot be analysed directly because it is processed during training, and it is not certain that it is well trained. Therefore, the analysis focus is decided to be on the *reward* function.

It is equally possible to substitute Equation (16) in Equation (15), such that the next state becomes a function completely dependent on the current state. In this way, both the vehicle and the action are considered, generating closed-loop dynamics. This can be appreciated in Equation (18).

$$s(t_k + 1) = f(s(t_k), \pi(s(t_k))) \tag{18}$$

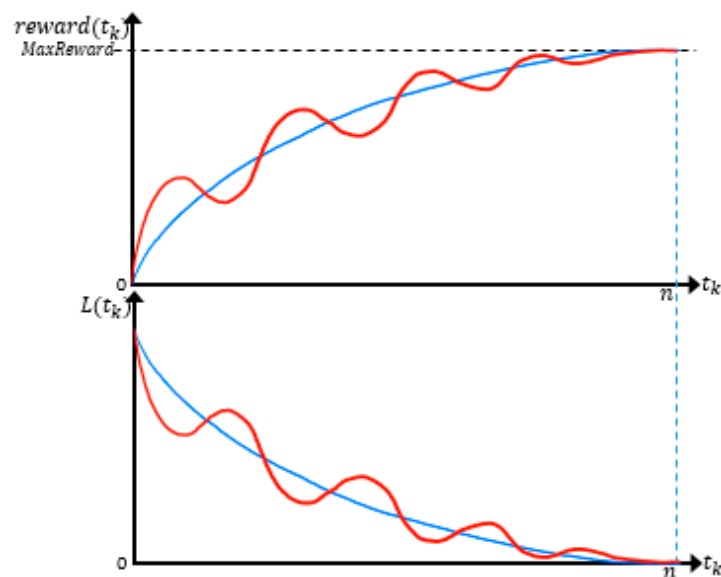
A maximum reward value is set, defined as *MaxReward*, which will have a value of 100. Therefore, the Lyapunov equation can be defined as shown in Equation (19). In this case,  $n$  [s] defines the *reward* measure in a future state. This equation defines a Lyapunov function because  $L$  is consistently positive. For the mathematical definition, see reference [42].

$$\begin{cases} L = \text{MaxReward} - \text{reward}(s(t_k)) > 0 \\ \Delta_n L = \text{reward}(s(t_k)) - \text{reward}(s(t_k + n)) < 0 \end{cases} \tag{19}$$

To explain the relationship that exists between the reward function and the Lyapunov function, Figure 8, which contains graphs by way of example, is presented. Paying attention only to the blue line, in the graph corresponding to the Lyapunov function, it can be seen that it converges. As the Lyapunov function tends to 0, the reward tends to the maximum reward value defined, as one decreases and the other grows. When this high value reward is obtained, it is interpreted that the vehicle has reached the target position. With reference to the red line, it must be taken into account that the reward not only evaluates the distance to the target but also the obstacles, so it can generate changes of direction. In this way, it is possible to ensure that the Lyapunov function will always be positive but needs analysis to check that its increment is negative. This provides a criterion for evaluating the suitability of the agent in terms of stability. Equation (19) is therefore evaluated cyclically.

The analysis will be performed with time steps in order to give Lyapunov time to demonstrate how it always decreases.

The number of time steps of Equation (19) is fixed in  $n = 200$  in order to measure the stability when the target point is reached. The intended objective, meaning to get out of the bounded zone, is not accomplished immediately, but in the future. Therefore,  $n$  is a parameter defined to be oriented to that future. After several tests, this constant is fixed, since in 200 steps the agent is able to trace a trajectory that manages to leave the limited area without collisions. The reason for not doing this instant by instant is that the path does not always have to be the optimal one, and, therefore, it does not have to improve its reward between  $t_k$  and  $t_k + 1$ , as can be seen in Figure 8. Instead of analysing the change of the Lyapunov function at each time step, it is proposed to guarantee the decrease of the Lyapunov function every  $n$  time steps. Therefore, it is suggested to examine only the decreases in the Lyapunov function over a long time. However, this does not imply that the agent does not encounter a stopping condition beforehand or even reach the target in less time.



**Figure 8.** Example graphs of the relationship between the reward value and the Lyapunov function.

Equation (19) is therefore verified for all possible  $s(0)$  within the navigation area, checking whether both conditions are fulfilled. It is always evaluated at  $s(0)$ , since the dynamics and the agent remain unchanged. Thus, the dynamics of the closed-loop system are predetermined, and different agents can be compared.

Therefore, this metric is suggested to establish whether the trajectory is stable, with the agent utilized as a controller.

### 3. Results

In order to demonstrate that the Lyapunov function remains a method for evaluating the adequacy of agents obtained through reinforcement learning, the process specified in Section 2 has been implemented. The results achieved in reference to the stability of the trajectories generated by two obtained agents are discussed. A comparison is made between the results of both to demonstrate that the Lyapunov function, in conjunction with the reward, can be used to evaluate the agents.

Two different training processes have been launched. These two training processes have some differences in the initial conditions. Firstly, as an initial starting condition for navigation, randomness has been established. This randomness has an impact on the starting position  $U_{1start}$  and  $U_{2start}$  of the autonomous vehicle, being variations that are always integers inside the navigation area, that is between 0 and 10 [m]. These variations have a randomness of 1. Such randomization has also been included in the navigation starting angle  $U_{3start}$ , with variances from 0 to  $2\pi$  [rad], with a distribution of 0.0001. Consequently, the training is not performed with the same initial conditions in the different iterations. Moreover, it should be noted that reinforcement learning through the actor proposes a different action in either case, so it is one more randomness component in reference to the training.

In addition, the MATLAB R2022a program is employed. This software includes a Toolbox dedicated to reinforcement learning, which gives the option to implement DDPG training. It is called a “Reinforcement Learning ToolBox” and enables the representation of policies and value functions using deep neural networks (see [43]). In order to be able to use this ToolBox, it is necessary to determine training parameters. To begin with, an end-of-training requirement of “Average Reward” equal to 200 was established. This value must be gained in a period of 50 iterations. These parameters are determined because it is considered that there are enough iterations to have obtained a successful result and that a satisfactory result with an average reward of 200 is sufficient for a good performance. For training, in addition, a maximum number of episodes is set to 20,000 and a maximum



Steps per episode value of 1000. The “verbose” parameter is also activated. These data are presented in Table 4.

**Table 4.** Neural networks training parameters.

Parameter Name	Value
Max Episodes	20,000
Max Steps Per Episode	1000
Score Averaging Window Length	50
Stop Training Criteria	Average Reward
Stop Training Value	200
Verbose	True

However, this does not mean that the best agent obtained in the training process is returned by the algorithm. The algorithm just returns the last agent trained at the final of the training process. The individual trained agents and their results are presented in different subsections.

It should be noted that the established sample time is  $T_{StepFunction} = 0.01$  [s]. Moreover, during the training of the agent, it takes an average of  $1.9922e^{-4}$  [s] to obtain the observation corresponding to the action proposed by the DDPG. Once the agent is trained and implemented in the computation of the Lyapunov function, it requires  $6.6e^{-3}$  [s] to return the generated movement after the initial position is given.

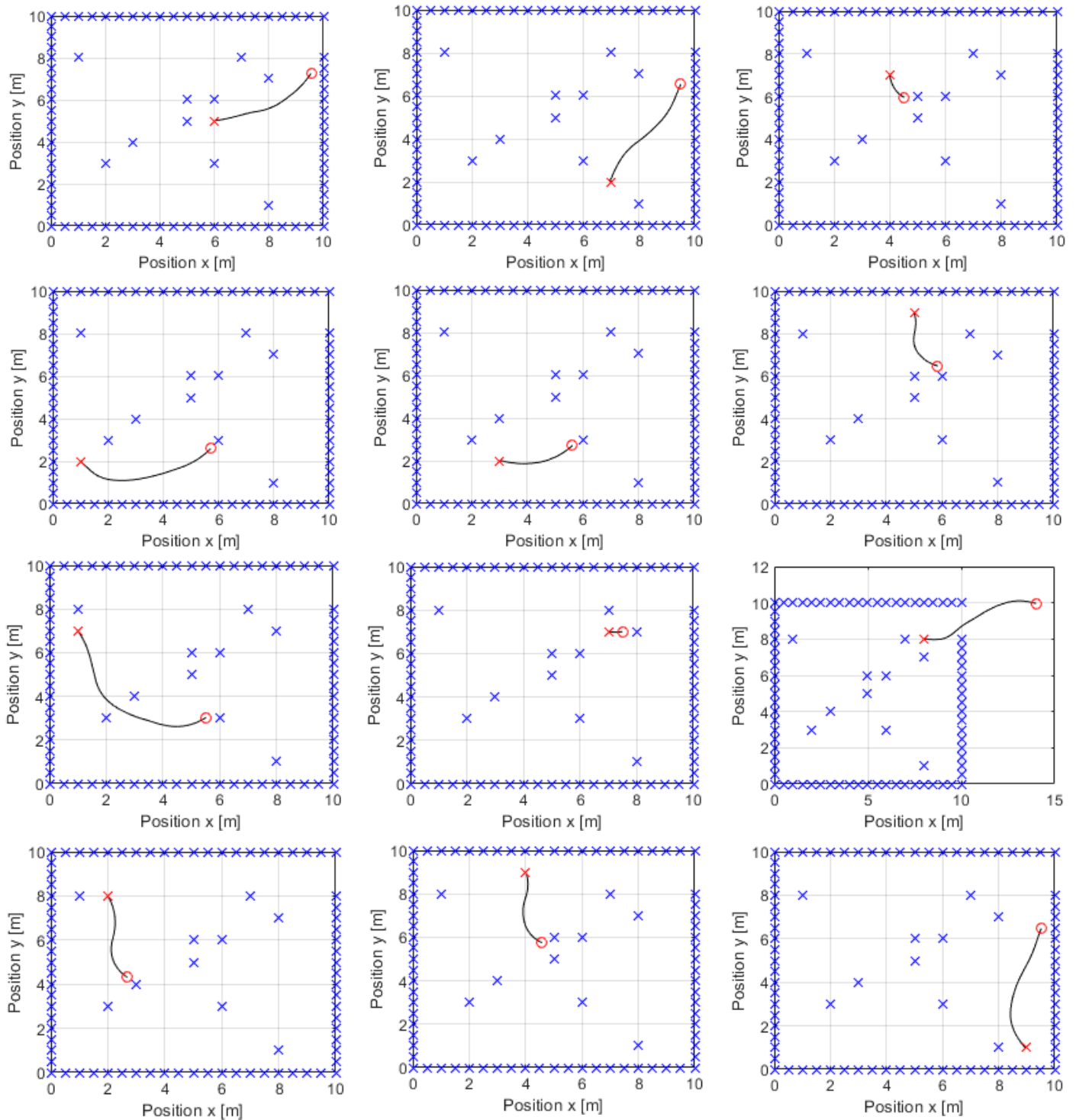
The two agents obtained through the different training pieces are called Agent K and Agent J for the purpose of presenting a clearer analysis.

### 3.1. Results Obtained for Agent K

Implementing the DDPG algorithm, an agent has been obtained, called Agent K, following the procedure presented in the flowchart shown in Figure 1. Agent K has been executed, and the trajectories it generates have been evaluated. For this purpose, the performance has been simulated starting from different origins, always from inside the navigation area defined in Figure 6. It is important to mention that the initial angle with which the vehicle starts navigation in all cases presented is 0 [rad]. The different trajectories generated by Agent K are shown in Figure 9, where the red circle indicates the maximum point that the AV can reach. At this red point, the navigation terminates. The black line represents the trajectory that the agent generates to try to reach the proposed objective.

As Figure 9 reveals, the trajectory proposed by Agent K on every occasion depends absolutely on the starting point of the navigation, as can be expected. However, it is noteworthy that when the trajectory encounters an obstacle, its experience makes it unable to avoid it and, instead, navigation fails. This refers to a lack of training. Nevertheless, the goal of the autonomous vehicle is to reach the destination, being this  $(x_{tarjet}, y_{tarjet}, \vartheta_{tarjet},) = [12, 12, 0]$ . Therefore, the analysis of these frames is focused on improving the position of the mobile robot on the future horizon.

Keeping this horizon in consideration, Agent K can navigate improving the distance to the target position. This is highly relevant because it is always well-directed and can correct the initial position in order to get closer to the destination. This means that there is an enhancement of the reward value in all navigation scenarios.



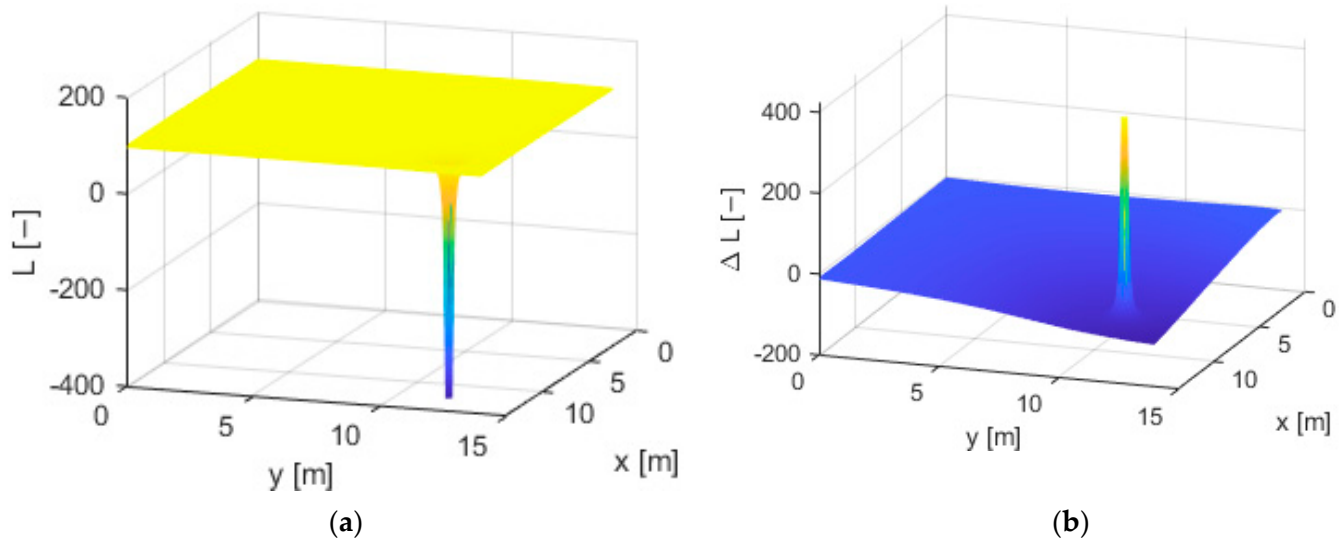
**Figure 9.** Different trajectories generated by Agent K, starting the navigation from different origins.

**Stability Analysis Results using Lyapunov’s Method for Agent K**

After checking the different trajectories that Agent K can generate, and assuming that it is constantly able to approach its destination, Equation (19) is implemented. For this section, the process explained in Figure 2 is followed. Referring to Equation (19), recall that it evaluates both the reward value that the agent obtains at its initial navigation point and the value at the future horizon.

Therefore, Equation (19) is applied, in which one part is the Lyapunov function and the other its increment. Recall that for a system to be stable, the Lyapunov equation must be

positive for all the positions, and its increment must be negative. An analysis of Equation 19 is carried out from all possible initial positions in order to verify the conditions of the Lyapunov function. In this way, the calculation is performed between the values 0 and 14 [m] of both axes, with a variation of 0.1 between each initial position. Hence, Agent K is tested, and the graphical results are presented in Figure 10. Figure 10a shows the surface of all the values of  $L$  obtained in the complete navigation area, while Figure 10b represents all the values of  $\Delta L$ .

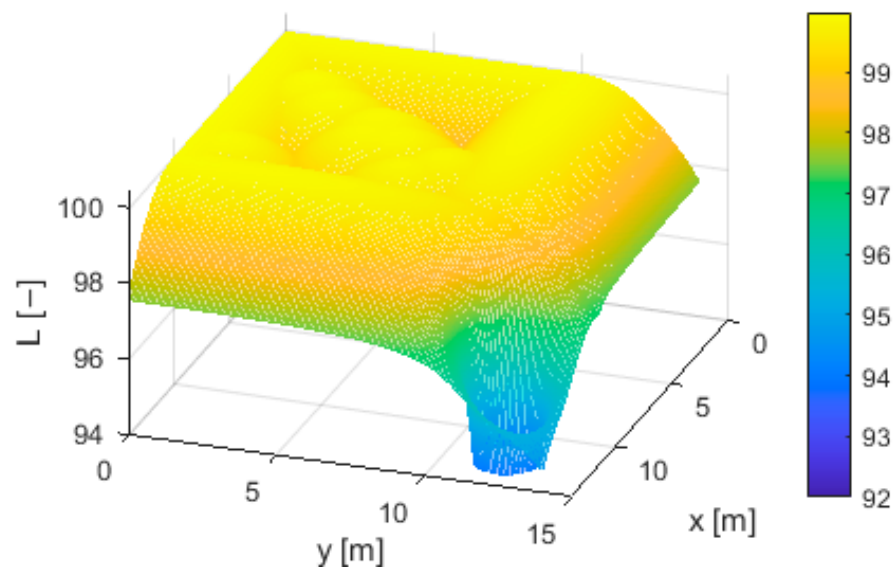


**Figure 10.** (a) Results of the Lyapunov function  $L$  in the entire navigation area for Agent K; (b) Results of the Lyapunov incremental function  $\Delta L$  over the entire navigation area for Agent K.

On the two surfaces of Figure 10, a peak can be appreciated, which is not of interest, since it causes the Lyapunov conditions not to be fulfilled. However, this peak occurs due to the term  $\left(\frac{1}{dist_{target}}\right)$  of the reward function (see Equation (13)). It can be identified that such a peak appears at the point of the map fixed as objective. Consequently, this peak is indifferent to the stability analysis that is being performed, since it is understood that it happens because an infinite reward value is obtained. So, it is obviated. In addition, the negative peak that appears in the function of  $L$ , appears inversely in  $\Delta L$ , being, respectively, a global minimum and global maximum. On these two surfaces in Figure 10, the stability conditions of the Lyapunov function can be verified. The Lyapunov function must be positive and its incremental function negative. As it can be seen, this is fulfilled.

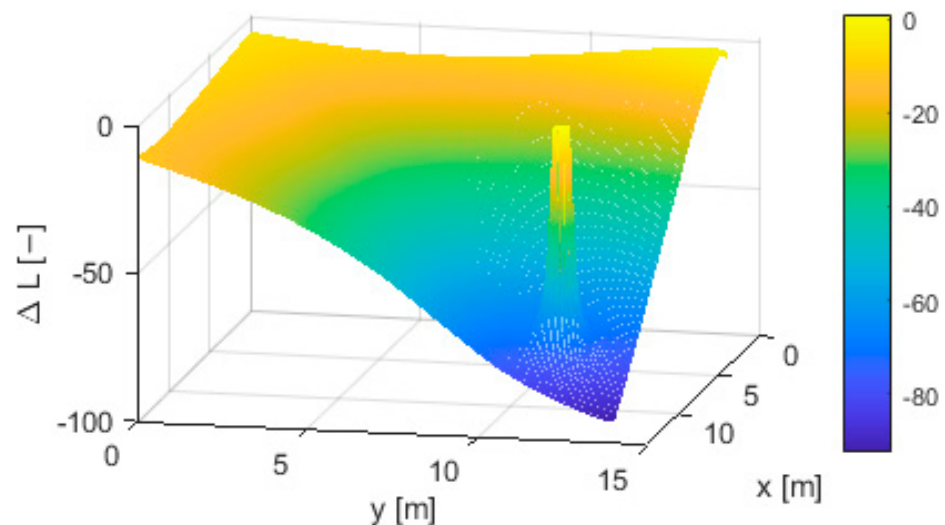
In order to perform a more accurate analysis of both plots, an enlargement of the sections of interest is realized. Thus, Figure 11 shows an enlargement of the area to be studied in Figure 10a.

The maximum value assumed by the Lyapunov function applied to all the points of the navigation area is  $L = 99.9862$ . Moreover, it can be observed that the whole area acquires positive values. It can be noted that local minimums are generated by the influence of the obstacles on the reward function. Therefore, it is possible to conclude that for  $L$  of Equation (19), the condition is fulfilled, that is, the Lyapunov function is positive in all positions.



**Figure 11.** Enlargement of the region of interest of the results of the Lyapunov function  $L$  in the whole navigation area for Agent K.

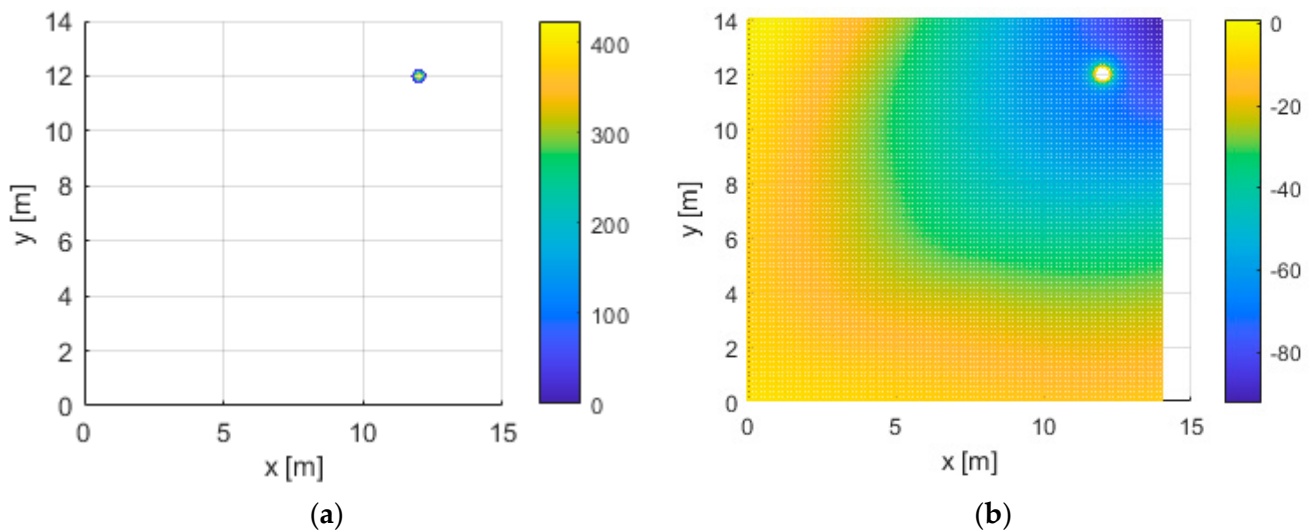
As in Figure 11, an enlargement of the area of interest of the  $\Delta L$  values surface (Figure 10b) is presented in Figure 12 for the purpose of analysis.



**Figure 12.** Interest zone enlargement of the results of the Lyapunov incremental function  $\Delta L$  over the entire navigation area for Agent K.

When calculating the Lyapunov increment, it can be appreciated how negative values are obtained in a future horizon. Moreover, a decreasing curve in the direction of the destination can be perceived. As a fact, the minimum value obtained is  $\Delta L = -92.4164$ . Therefore, the fulfilment of the Lyapunov incremental function can be assured, that is, the second part of Equation (19) is negative for all positions in the study area.

As the Lyapunov incremental function is the one that may cause more uncertainty due to its values, which get closer to 0 at the edges of the navigation area; Figure 13 is presented. Figure 13 is intended to show the values by ranges. Figure 13a shows the values obtained by the Lyapunov increment in a range between 0 and infinity. On the other hand, Figure 13b shows the values of the same function but between 0 and minus infinity.



**Figure 13.** (a) Area of positive values obtained from the Lyapunov incremental function  $\Delta L$  over the entire navigation surface for Agent K; (b) Area of negative values obtained from the Lyapunov incremental function  $\Delta L$  over the entire navigation surface for Agent K.

Figure 13a reveals that the positive values generated by Agent K when tested under Equation (19) are those of the peak with a tendency to infinity. However, Figure 13b demonstrates what is actually of interest. All the values of the navigation area for a future horizon adopt a negative value guaranteeing the accomplishment of the requirement.

In this way, it is demonstrated that Agent K generates stable systems since the requirements of Equation (19) are fulfilled. It should be remembered that this achievement is performed in a future horizon, focused on reaching the fixed positioning objective. However, Agent K fails to accomplish any of the navigation objectives.

Therefore, it has been possible to examine the stability of the trajectories generated by Agent K through its reward function applied in the Lyapunov function. In this way, it is possible to evaluate the satisfaction with Agent K in the application of the navigation of the study.

### 3.2. Results Obtained for Agent J

With the purpose of demonstrating that it is possible to use the Lyapunov function together with the reward as a metric to evaluate if the agent obtained through DDPG is adequate for the application of AV navigation, the results of a new experiment are presented. In order to compare the results, training has been carried out and, therefore, a new agent, called Agent J, has been obtained.

As done with Agent K, Agent J is tested through its experience buffer obtained from training. For a more appropriate comparison with respect to the analysis of Agent K, Agent J is subjected to the generation of trajectories from the same starting points as in Figure 9. Figure 14 is therefore presented where the different trajectories generated by Agent J are plotted.



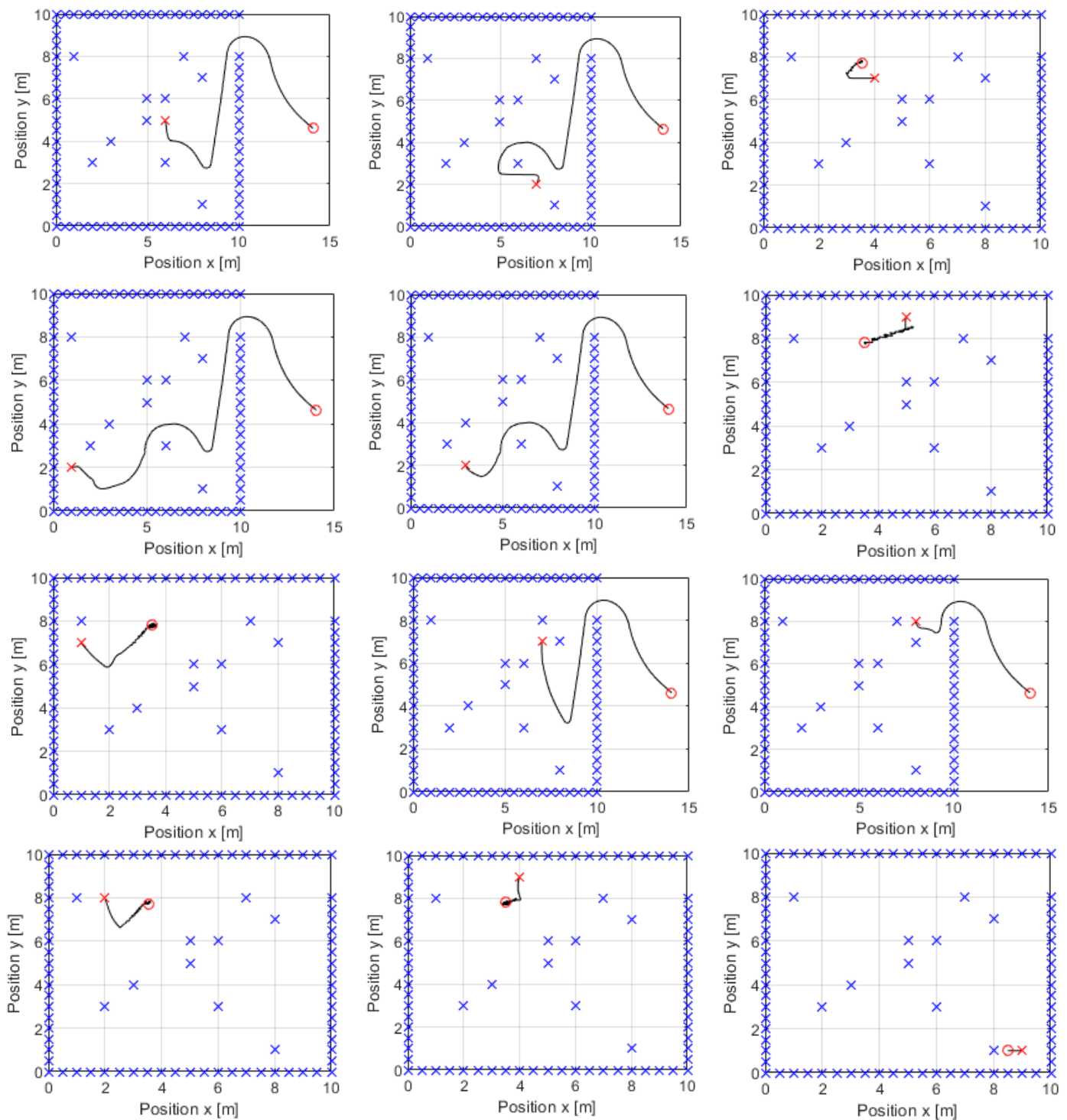


Figure 14. Different trajectories generated by Agent J, starting the navigation from different origins.

On this occasion, it is noticeable that Agent J can generate longer trajectories, practically reaching the destination given in  $(x_{tarjet}, y_{tarjet}, \vartheta_{tarjet}) = [12, 12, 0]$ . These paths in many cases are even able to leave the region bounded by the margins. Moreover, it is additionally able to avoid obstacles. With these data, Agent J gives better results in reference to the fulfilment of the navigation objectives.

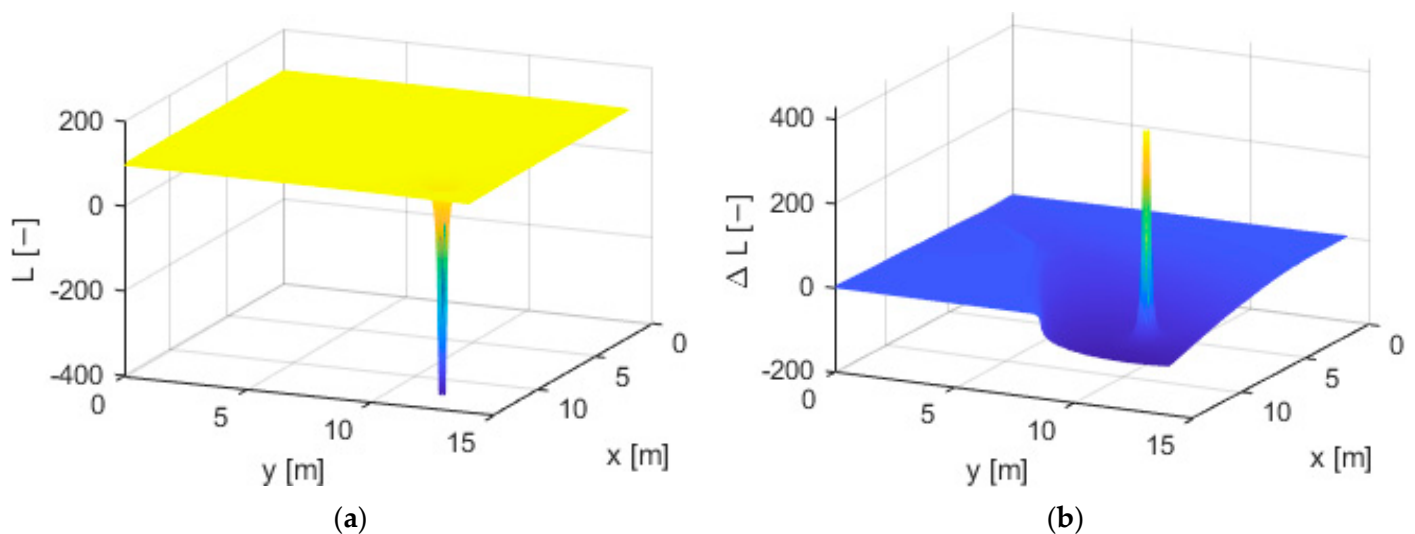
It can be appreciated that the generated trajectories by Agent J are not optimal. Furthermore, note that some of the generated trajectories are abrupt. As an initial evaluation, it can be concluded that not all the trajectories that Agent J generates improve the position



in a given horizon, being unable to ensure an increase of the reward value between the initial and final position.

#### Stability Analysis Results Using Lyapunov's Method for Agent J

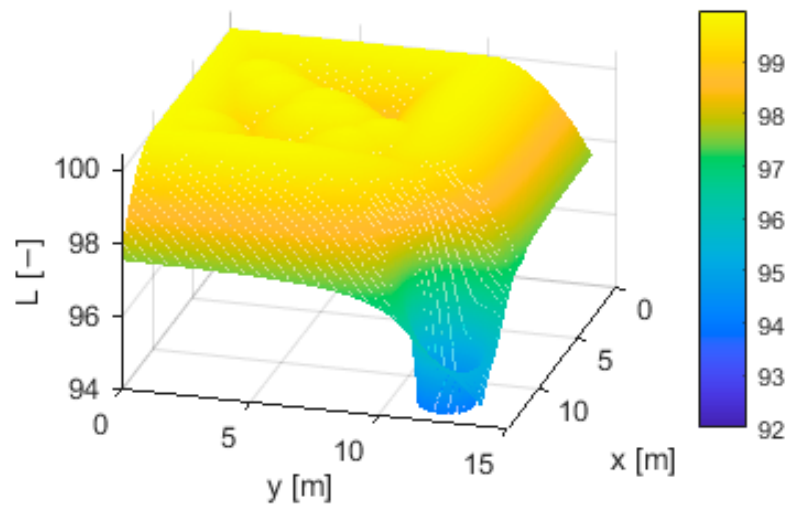
After checking the most appreciable discrepancies of the trajectories generated by Agent J, Equation (19) is applied according to the procedure explained in Figure 2. For the system to be stable and able to prove that the agent is appropriate, the Lyapunov function must be positive while the increment of the Lyapunov function must be negative. Once Agent J has been tested, the results obtained are presented in Figure 15. The left figure, Figure 15a, plots the values of  $L$  in the navigation area, while the right figure, Figure 15b, shows the values obtained for  $\Delta L$ .



**Figure 15.** (a) Results of the Lyapunov function  $L$  in the entire navigation area for Agent J; (b) Results of the Lyapunov incremental function  $\Delta L$  over the entire navigation area for Agent J.

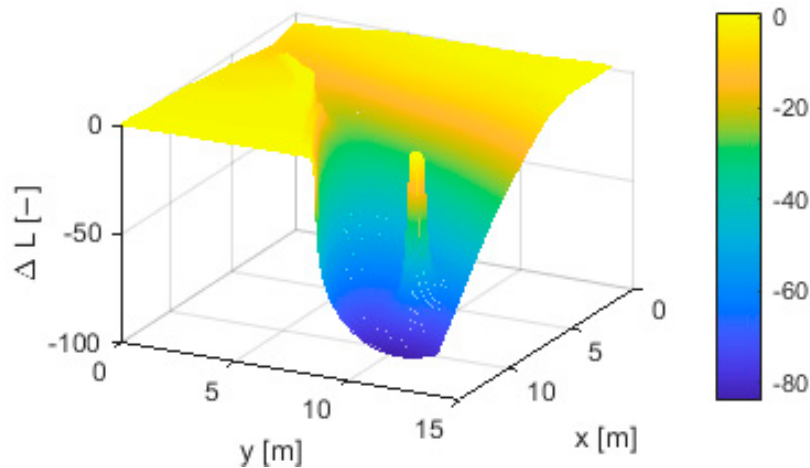
As observed in the analysis of Agent K, peaks appear again on both surfaces, as can be seen in Figure 15. As shown in Figure 15b,  $\Delta L$  functions peak is positive, being a global maximum. This peak is negative on the surface, which refers to  $L$ , representing a global minimum. As for an overview, the Lyapunov function obtains positive values in its entirety, but its increment may not, as it appreciates that it is close to positive values, despite being small values.

In order to perform a more precise analysis, an enlargement of Figure 15a is shown in Figure 16. It can be appreciated that the values of the Lyapunov function are positive for the whole navigation area for Agent J. The maximum value obtained by this function for Agent J is  $L = 99.9826$ . In addition, as for Agent K, the obstacles generate an influence on the calculation of the Lyapunov function, generating local minimums. Therefore, it can be assumed that for  $L$ , the condition of obtaining positive values in all positions, with an established future horizon, is fulfilled.



**Figure 16.** Enlargement of the region of interest of the results of the Lyapunov function  $L$  over the whole navigation area for Agent J.

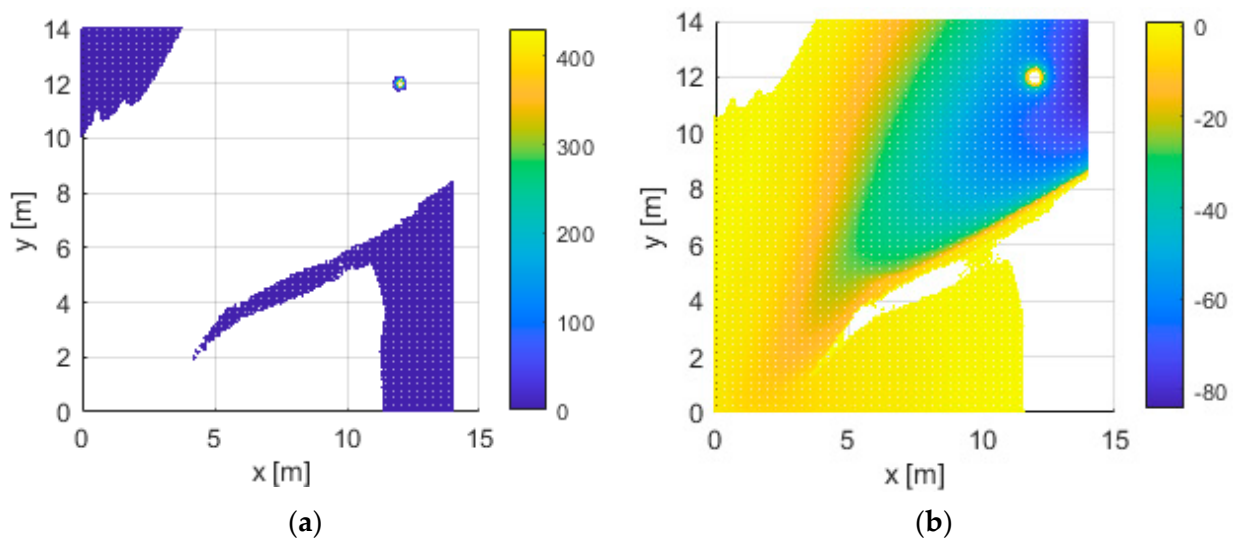
In the same manner, as in Figure 16, there is also an enlargement of the surface plot of the Lyapunov increment function shown in Figure 15b. This enlargement is shown in Figure 17, where the area to be analysed can be appreciated.



**Figure 17.** Interest zone enlargement of the results of the Lyapunov increment function  $\Delta L$  over the entire navigation area Agent J.

When calculating  $\Delta L$  for a future horizon, most of the positions in Figure 17 assume negative values, although some zones close to zero obtain small positive values. These zones correspond to the corners of the study area. In turn, the values have a decreasing tendency as they approach the point established as the target. The minimum value obtained for Agent J is  $\Delta L = -84.0685$ .

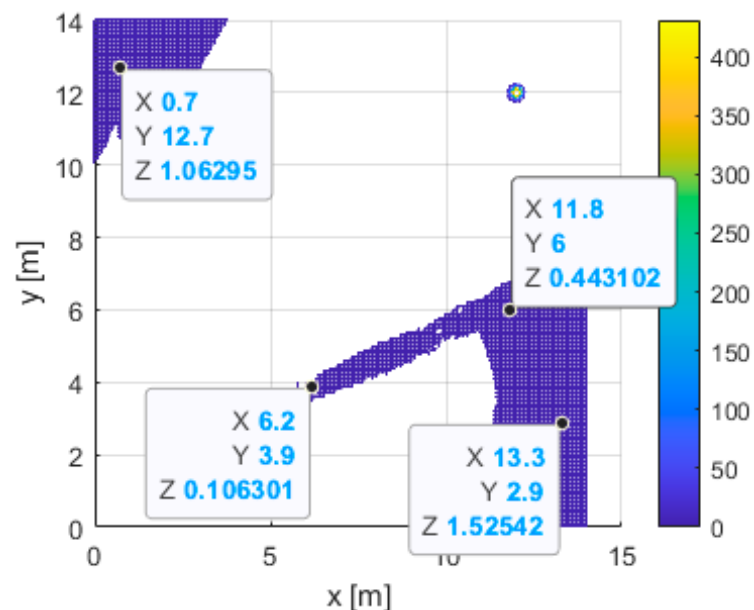
To visualise the positive values obtained, Figure 18 is presented. Figure 18, as was done with Figure 13, shows the values by ranges. Figure 18a shows the Lyapunov incremental function in the range from 0 to infinity, while Figure 18b shows the Lyapunov incremental function in the range from 0 to minus infinity.



**Figure 18.** (a) Area of positive values obtained from the Lyapunov incremental function  $\Delta L$  over the entire navigation surface for Agent J.; (b) Area of negative values obtained from the Lyapunov incremental function  $\Delta L$  over the entire navigation surface for Agent J.

Figure 18b illustrates what was expected, that is, that the values at a given horizon would adopt negative values throughout the delimited area. However, some positions do not have a negative value. Figure 18a demonstrates that the value of the Lyapunov increment at these positions is positive.

Focusing now on the analysis of Figure 18a, there are positions that assume positive values that do not fulfil the conditions previously established in Equation (19). This is prejudicial to the stability, since one of the conditions of the Lyapunov function is not satisfied. It can be noticed that the zones with positive values appear mainly at the edges of the delimited area, but there are also positive values in a central zone. In order to give a numerical value to these positive areas, Figure 19 is presented, where labels have been added to appreciate the value obtained by  $\Delta L$ .



**Figure 19.** Data of the values outside the desired range obtained through the Lyapunov incremental function  $\Delta L$  for Agent J.

In Figure 19, the labels X and Y values represent the position on the map over which the Lyapunov incremental function has been measured. Thus, Z indicates the numerical value of  $\Delta L$ . It can be observed that in the positive central areas the value of  $\Delta L$  is relatively small, but not so at the edges, where it exceeds the value of 1. This is a problem, since it is not possible to guarantee stability.

Consequently, it cannot be assured that Agent J is suitable for the application of navigation in this research area. The study of the Lyapunov function has concluded that it does not generate stable systems and is, therefore, not a suitable agent in future horizons to accomplish the function.

### 3.3. Comparison of Results for Agent K and Agent J

Having carried out the analysis of the trajectories proposed by Agent K and Agent J for the navigation of the application in Figure 6, several points can be highlighted.

Regarding the achievement of navigation objectives, it could be observed that Agent K does not manage to avoid obstacles and encounters navigation termination conditions. In addition, Agent K is only able to generate trajectories that leave the bounded zone if the navigation starts near the exit of the area. With reference to Agent J, it manages both to avoid obstacles and to leave the navigation area numerous times, fulfilling both objectives of navigation control. However, there are areas from which navigation is initiated, and from which it does not know how to generate a correct trajectory.

In terms of the study of stability through the Lyapunov function, it has been shown that Agent K as a controller generates stable systems. The same is not true for Agent J as the conditions of the Lyapunov function are not fulfilled.

By focusing exclusively on the analysis of the achievement of navigation goals, Agent J is found to generate better results and would therefore lead to the selection of this agent as the controller of the autonomous vehicle. However, using the Lyapunov function with the reward function as a tool to assess the suitability of this agent, it is found that it does not comply with the stability conditions. Therefore, this agent would be unchosen for the navigation application. Agent K could be selected as it always generates stable trajectories, but it would require running the controller numerous times. Overall, it can be concluded that both agents need to be further trained.

Therefore, by detecting which are the zones in the navigation area where the Lyapunov function does not decrease, it is possible to detect zones that should be worked on in future training sessions. In this way, it can be evaluated whether the agent obtained through the DDPG method is suitable for the desired application.

## 4. Discussion and Conclusions

According to previous studies, and as described in the introduction, on many occasions the DDPG method produces agents that can generate trajectories that avoid obstacles but do not always guarantee the arrival of the autonomous vehicle at the target. Moreover, this last objective is always reached in a future horizon, after several iterations of the control algorithm. Therefore, a fusion of different techniques [28] tends to be performed in order to accomplish all the proposed navigation objectives. There is no way to recognize why the agent, as a controller, is not able to fulfil the proposed navigation goals. Thus, it is not possible to evaluate the results before performing the combination of techniques.

Usually, the problem resides in the definition of the reward function [25,26] because of the multiple variables that can be considered in it. In addition, it is important to consider that different navigation areas have many similarities, and, therefore, it is not guaranteed that the movements would be optimal. Therefore, the reward function is not something generic and independent of the desired navigation application.

Furthermore, instead of focusing on the appropriateness of the obtained agent, there is a tendency to add parameters to the reward function, or even to explore different ways of evaluating it [24,27]. It is not considered a bad option to convert the reward function into a more specific one. Moreover, since the agent generates experience buffers, these

can be accumulated [29] and added to the reward evaluation. However, this still does not guarantee the agent to be obtained through the DDPG is adequate in its entirety.

In general, previous studies do not evaluate the agents obtained from the DDPG as there is no method of verifying their adequacy and the agent that provides the best results in terms of achieving the navigation objectives is selected. It has been demonstrated that this is not the most accurate practice.

In this paper, it has been presented that there is a mode to evaluate the suitability of an agent obtained through the Deep Deterministic Policy Gradient algorithm. This evaluation is done through the Lyapunov function, based on the reward used for the DDPG training. Rather than a way of solving the problem of navigation or even deficiencies in reinforcement learning training, this technique remains a method of evaluating the solutions obtained.

When an agent is obtained through the DDPG method, the agent's experience buffer can generate trajectories. These paths can comply with all the navigation objectives set for the task. However, it is possible to obtain agents that generate trajectories that do not satisfy the navigation objectives set.

Such agents, through the reward function by which they have been acquired, can be evaluated in terms of stability. This stability is assessed in future horizons by means of the trajectories that the agents generate. Therefore, it can be seen whether, in a navigation zone, the system generated by the controller to be used is stable. In this way, an evaluation can be made of the satisfaction with that agent and whether it is applicable to the required task.

In addition, the analysis proposed in this article also provides a useful indication of which parts of the navigation area are deficient in terms of training.

Regarding future lines, the next phase of the project would be the use of the agent as a controller in a real implementation of an autonomous vehicle. Using computer vision, it is possible to detect a navigation surface that can be included as a condition in the DDPG training and verify the trajectories proposed by the agent, having previously checked the stability of these trajectories in a future horizon. Moreover, as it has been appreciated, this method does not generate optimal trajectories, so a study could also be carried out on it.

The proposed neural networks may not be suitable for the proposed application, so an analysis might be performed on them and find out why the training does not return the expected results. Besides, the inclusion of genetic algorithms in the training of the DDPG can be suggested as future research.

**Author Contributions:** Conceptualization, M.C.-O. and E.Z.; methodology, M.C.-O.; software, M.C.-O. and E.Z.; validation, E.Z. and U.F.-G.; formal analysis, M.C.-O., E.Z. and A.T.-F.-B.; investigation, M.C.-O. and A.S.-C.; resources, U.F.-G.; writing—original draft preparation, M.C.-O. and E.Z.; writing—review and editing, U.F.-G.; All authors have read and agreed to the published version of the manuscript.

**Funding:** The current study has been sponsored by the Government of the Basque Country-ELKARTEK21/10 KK-2021/00014 research program "Estudio de nuevas técnicas de inteligencia artificial basadas en Deep Learning dirigidas a la optimización de procesos industriales".

**Data Availability Statement:** Data will be available upon reasonable request to the corresponding author.

**Acknowledgments:** The authors are grateful for the support provided by UPV/EHU.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Xie, L.; Scheifele, C.; Xu, W.; Stol, K.A. Heavy-duty omni-directional Mecanum-wheeled robot for autonomous navigation: System development and simulation realization. In Proceedings of the 015 IEEE International Conference on Mechatronics (ICM), Nagoya, Japan, 6–8 March 2015; pp. 256–261. [\[CrossRef\]](#)
2. Piemngam, K.; Nilkhamhang, I.; Bunnun, P. Development of Autonomous Mobile Robot Platform with Mecanum Wheels. In Proceedings of the 2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP), Bangkok, Thailand, 16–18 January 2019; pp. 90–93. [\[CrossRef\]](#)



3. Kim, J.; Woo, S.; Kim, J.; Do, J.; Kim, S.; Bae, S. Inertial navigation system for an automatic guided vehicle with Mecanum wheels. *Int. J. Precis. Eng. Manuf.* **2012**, *13*, 379–386. [[CrossRef](#)]
4. Li, Y.; Dai, S.; Shi, Y.; Zhao, L.; Ding, M. Navigation Simulation of a Mecanum Wheel Mobile Robot Based on an Improved A\* Algorithm in Unity3D. *Sensors* **2019**, *19*, 2976. [[CrossRef](#)] [[PubMed](#)]
5. Liu, L.S.; Lin, J.F.; Yao, J.X.; He, D.W.; Zheng, J.S.; Huang, J.; Shi, P. Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach. *Wirel. Commun. Mob. Comput.* **2021**, *2021*, 8881684. [[CrossRef](#)]
6. Fox, D.; Burgard, W.; Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robot. Autom. Mag.* **1997**, *4*, 23–33. [[CrossRef](#)]
7. Brock, O. High-speed Navigation Using the Global Dynamic Window Approach. In Proceedings of the Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), Detroit, MI, USA, 10–15 May 1999. [[CrossRef](#)]
8. Saranrittichai, P.; Niparnan, N.; Sudsang, A. Robust local obstacle avoidance for mobile robot based on Dynamic Window approach. In Proceedings of the 2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Krabi, Thailand, 15–17 May 2013. [[CrossRef](#)]
9. Xie, L.; Henkel, C.; Stol, K.; Xu, W. Power-minimization and energy-reduction autonomous navigation of an omnidirectional Mecanum robot via the dynamic window approach local trajectory planning. *Int. J. Adv. Robot. Syst.* **2018**, *15*, 1–12. [[CrossRef](#)]
10. Borenstein, J.; Koren, Y. The Vector Field Histogram-fast obstacle avoidance for mobile robots. *IEEE J. Robot. Autom.* **1991**, *7*, 278–288. [[CrossRef](#)]
11. Ye, C. Navigating a mobile robot by a traversability field histogram. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **2007**, *37*, 361–372. [[CrossRef](#)]
12. Burgos, E.; Bhandari, S. Potential flow field navigation with virtual force field for UAS collision avoidance. In Proceedings of the 2016 International Conference on Unmanned Aircraft Systems (ICUAS), Arlington, VA, USA, 7–10 June 2016; pp. 505–513. [[CrossRef](#)]
13. Wang, T.; Yan, X.; Wang, Y.; Wu, Q. A distributed model predictive control using virtual field force for multi-ship collision avoidance under COLREGs. In Proceedings of the 2017 4th International Conference on Transportation Information and Safety (ICTIS), Banff, AB, Canada, 8–10 August 2017; pp. 296–305. [[CrossRef](#)]
14. Maarif, A.; Rahmani, W.; Vera, M.A.M.; Nuryono, A.A.; Majdoubi, R.; Cakan, A. Artificial Potential Field Algorithm for Obstacle Avoidance in UAV Quadrotor for Dynamic Environment. In Proceedings of the 2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT), Purwokerto, Indonesia, 17–18 July 2021; pp. 184–189. [[CrossRef](#)]
15. Hoffmann, G.M.; Tomlin, C.J.; Montemerlo, M.; Thrun, S. Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing. In Proceedings of the 2007 American Control Conference, New York, NY, USA, 9–13 July 2007; pp. 2296–2301. [[CrossRef](#)]
16. Cabezas-Olivenza, M.; Zulueta, E.; Sanchez-Chica, A.; Teso-fz-Betoño, A.; Fernandez-Gamiz, U. Dynamical Analysis of a Navigation Algorithm. *Mathematics* **2021**, *9*, 3139. [[CrossRef](#)]
17. AbdElmoniem, A.; Osama, A.; Abdelaziz, M.; Maged, S.A. A path-tracking algorithm using predictive Stanley lateral controller. *Int. J. Adv. Robot. Syst.* **2020**, *17*, 1–11. [[CrossRef](#)]
18. Missura, M.; Bennewitz, M. Predictive collision avoidance for the dynamic window approach. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8620–8626. [[CrossRef](#)]
19. Ögren, P.; Leonard, N.E. A convergent dynamic window approach to obstacle avoidance. *IEEE Trans. Robot.* **2005**, *21*, 188–195. [[CrossRef](#)]
20. Kashyap, A.K.; Parhi, D.R.; Muni, M.K.; Pandey, K.K. A hybrid technique for path planning of humanoid robot NAO in static and dynamic terrains. *Appl. Soft Comput. J.* **2020**, *96*, 106581. [[CrossRef](#)]
21. Dobrevski, M.; Skocaj, D. Adaptive dynamic window approach for local navigation. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; pp. 6930–6936. [[CrossRef](#)]
22. Chang, L.; Shan, L.; Jiang, C.; Dai, Y. Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment. *Auton. Robots* **2021**, *45*, 51–76. [[CrossRef](#)]
23. Bouhamed, O.; Ghazzai, H.; Besbes, H.; Massoud, Y. Autonomous UAV navigation: A DDPG-based deep reinforcement learning approach. In Proceedings of the 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 12–14 October 2020. [[CrossRef](#)]
24. You, S.; Diao, M.; Gao, L.; Zhang, F.; Wang, H. Target tracking strategy using deep deterministic policy gradient. *Appl. Soft Comput.* **2020**, *95*, 106490. [[CrossRef](#)]
25. Zhang, D.; Bailey, C.P. Obstacle avoidance and navigation utilizing reinforcement learning with reward shaping. *Proc. SPIE Artif. Intell. Mach. Learn. Multi-Domain Oper. Appl. II* **2020**, *11413*, 114131H. [[CrossRef](#)]
26. Xie, L.; Miao, Y.; Wang, S.; Blunsom, P.; Wang, Z.; Chen, C.; Markham, A.; Trigoni, N. Learning with Stochastic Guidance for Robot Navigation. *IEEE Trans. Neural Networks Learn. Syst.* **2021**, *32*, 166–176. [[CrossRef](#)]
27. Gao, X.; Yan, L.; Wang, G.; Wang, T.; Du, N.; Gerada, C. Toward Obstacle Avoidance for Mobile Robots Using Deep Reinforcement Learning Algorithm. In Proceedings of the 2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA), Chengdu, China, 1–4 August 2021; pp. 2136–2139. [[CrossRef](#)]



28. Bouhamed, O.; Ghazzai, H.; Besbes, H.; Massoud, Y. A UAV-Assisted Data Collection for Wireless Sensor Networks: Autonomous Navigation and Scheduling. *IEEE Access* **2020**, *8*, 110446–110460. [[CrossRef](#)]
29. Guo, S.; Zhang, X.; Zheng, Y.; Du, Y. An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning. *Sensors* **2020**, *20*, 426. [[CrossRef](#)]
30. Liu, M.; Zhao, F.; Niu, J.; Liu, Y. Reinforcement Driving: Exploring Trajectories and Navigation for Autonomous Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 808–820. [[CrossRef](#)]
31. Matheron, G.; Perrin, N.; Sigaud, O. Understanding Failures of Deterministic Actor-Critic with Continuous Action Spaces and Sparse Rewards. In Proceedings of the 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, 15–18 September 2020. [[CrossRef](#)]
32. Grando, R.B.; de Jesus, J.C.; Kich, V.A.; Kolling, A.H.; Guerra, R.S.; Drews-Jr, P.L.J. Deterministic and Stochastic Analysis of Deep Reinforcement Learning for Low Dimensional Sensing-Based Navigation of Mobile Robots. *arXiv* **2022**, arXiv:2209.06328.
33. Kormushev, P.; Calinon, S.; Caldwell, D.G. Imitation Learning of Positional and Force Skills Demonstrated via Kinesthetic Teaching and Haptic Input. *Adv. Robot.* **2011**, *25*, 581–603. [[CrossRef](#)]
34. Hussein, A.; Elyan, E.; Gaber, M.M.; Jayne, C. Deep Imitation Learning for 3D Navigation Tasks. *Neural Comput. Appl.* **2018**, *29*, 389–404. [[CrossRef](#)]
35. Tesfazgi, S.; Lederer, A.; Hirche, S. Inverse Reinforcement Learning: A Control Lyapunov Approach. In Proceedings of the 2021 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA, 14–17 December 2021; pp. 3627–3632. [[CrossRef](#)]
36. Choi, S.; Kim, S.; Kim, H.J. Inverse Reinforcement Learning Control for Trajectory Tracking of a Multicopter UAV. *Int. J. Control Autom. Syst.* **2017**, *15*, 1826–1834. [[CrossRef](#)]
37. Graña, M. Basque Conference on Cyber Physical Systems and Artificial Intelligence. In Proceedings of the Basque Conference on Cyber Physical Systems and Artificial Intelligence, San Sebastian, Spain, 18–19 May 2022. [[CrossRef](#)]
38. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**. [[CrossRef](#)]
39. La Salle, J.; Lefschetz, S.; Alverson, R.C. Stability by Liapunov's Direct Method With Applications. *Phys. Today* **1962**, *15*, 59. [[CrossRef](#)]
40. Bailey, F.N. The Application of Lyapunov's Second Method to Interconnected Systems. *J. Soc. Ind. Appl. Math. Ser. A Control* **1965**, *3*, 443–462. [[CrossRef](#)]
41. Kalman, R.E.; Bertram, J.E. Control system analysis and design via the second method of lyapunov: (I) continuous-time systems (II) discrete time systems. *IRE Trans. Autom. Control* **1959**, *4*, 112. [[CrossRef](#)]
42. Keller-Ressel, M.; Lyapunov Function. From MathWorld—A Wolfram Web Resource, Created by Eric W. Weisstein. Available online: <https://mathworld.wolfram.com/LyapunovFunction.html> (accessed on 21 November 2022).
43. Reinforcement Learning Toolbox—MATLAB. Available online: <https://es.mathworks.com/products/reinforcement-learning.html> (accessed on 21 November 2022).

**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.