applied sciences

MDPI

Article

Trajectory Planning for a Mobile Robot in a Dynamic Environment Using an LSTM Neural Network

Alejandra Molina-Leal [1], Alfonso Gómez-Espinosa [1,*], Jesús Arturo Escobedo Cabello [1], Enrique Cuan-Urquizo [1] and Sergio R. Cruz-Ramírez [2]

[1] Tecnologico de Monterrey, Escuela de Ingenieria y Ciencias, Av. Epigmenio González 500, Fracc. San Pablo, Querétaro 76130, Mexico; amolina@tec.mx (A.M.-L.); arturo.escobedo@tec.mx (J.A.E.C.); ecuanurqui@tec.mx (E.C.-U.)

[2] Tecnologico de Monterrey, Escuela de Ingenieria y Ciencias, Av. Eugenio Garza Sada 300, Lomas del Tecnológico, San Luis Potosí 78211, Mexico; rolando.cruz@tec.mx

* Correspondence: agomeze@tec.mx; Tel.: +52-442-238-3302

Abstract: Autonomous mobile robots are an important focus of current research due to the advantages they bring to the industry, such as performing dangerous tasks with greater precision than humans. An autonomous mobile robot must be able to generate a collision-free trajectory while avoiding static and dynamic obstacles from the specified start location to the target location. Machine learning, a sub-field of artificial intelligence, is applied to create a Long Short-Term Memory (LSTM) neural network that is implemented and executed to allow a mobile robot to find the trajectory between two points and navigate while avoiding a dynamic obstacle. The input of the network is the distance between the mobile robot and the obstacles thrown by the LiDAR sensor, the desired target location, and the mobile robot's location with respect to the odometry reference frame. Using the model to learn the mapping between input and output in the sample data, the linear and angular velocity of the mobile robot are obtained. The mobile robot and its dynamic environment are simulated in Gazebo, which is an open-source 3D robotics simulator. Gazebo can be synchronized with ROS (Robot Operating System). The computational experiments show that the network model can plan a safe navigation path in a dynamic environment. The best test accuracy obtained was 99.24%, where the model can generalize other trajectories for which it was not specifically trained within a 15 cm radius of a trained destination position.

Keywords: mobile robot; obstacle avoidance; LSTM neural network; dynamic path planning

Citation: Molina-Leal, A.; Gómez-Espinosa, A.; Escobedo Cabello, J.A.; Cuan-Urquizo, E.; Cruz-Ramírez, S.R. Trajectory Planning for a Mobile Robot in a Dynamic Environment Using an LSTM Neural Network. *Appl. Sci.* **2021**, *11*, 10689. https://doi.org/10.3390/app112210689

Academic Editor: Nicola Pio Belfiore

Received: 21 September 2021
Accepted: 2 November 2021
Published: 12 November 2021

1. Introduction

Over the last three decades, mobile robotics research has been a prominent topic. One of the most basic problems in mobile robotics is obstacle avoidance [1]. Because of this, trajectory planning is an essential task for the autonomous mobile robot; it is desired to find a collision-free motion in an obstacle prone environment to achieve a safe navigation path from an initial location to an end location [2].

Machine learning, an application of artificial intelligence (AI), has been widely used for robotic trajectory generation and navigation. The main purpose of machine learning is to allow a system to learn without the need to automatically and constantly program it. Hand-programmed algorithms may not be the best learning system for an autonomous robot in the real world because of the continuously changing environment and the uncertainty caused by these changes. P. Ehlert [2] affirms that an artificial neural network (ANN) provides an adaptable learning system. The main advantage of using neural networks is the fact that the system does not require specific properties for specific issues, the system determines these properties on its own, and the mobile robot operator only provides the system with training examples and the corresponding action or reinforcement.

A mobile robot is a highly nonlinear system [3]. Mobile robots are known to be nonholonomic, only moving normally following the axis of the wheels. For these reasons, it is arduous to design and implement an autonomous trajectory planning mobile robot.

Different AI approaches for mobile robot trajectory planning and navigation have been proposed throughout the last decades. A. Maw et al. [4] proposed a hybrid path planning algorithm that uses an anytime graph-based path planning algorithm for global planning and deep reinforcement learning for local planning, which is applied for a real-time mission planning system of an autonomous UAV that is adaptive to real-world environments consisting of both static and moving obstacles. Al-Taharwa et al. [5] presented a mobile robot path planning genetic algorithm (GA) in a static environment with a simplified fitness function that employs the path length. The authors demonstrated that the proposed algorithm was efficient in handling different types of tasks in static environments. A. Bakdi et al. [6] developed a GA to generate a collision-free optimal path linking an initial configuration of the mobile robot to a final configuration, and presented an adaptive fuzzy-logic controller to keep track of the robot on the desired path. U. Orozco-Rosas et al. [7] suggested a membrane evolutionary artificial potential field (memEAPF) approach combining membrane computing with a GA and the artificial potential field (APF) method to find the parameters to generate a safe trajectory for the mobile robot. E. Low et al. [8] used improved Q-learning, a type of reinforcement learning, with the flower pollination algorithm (FPA) to find the optimal static trajectory planning of a mobile robot. The authors stated that an ANN-based algorithm could be integrated, since it has shown good results in processing spacious states and actions in trajectory planning [8,9].

Regarding the use of neural networks as a solution to this problem, other approaches have also been carried out. N. Noguchi and H. Terao [10] proposed a neural network (NN) to represent the dynamic model of an agricultural mobile robot. They trained the NN with a backpropagation (BP) algorithm that was accurate in simulating the mobile robot path. The inputs of the NN were the time series of the steer angles and the changes in the steer angles of the mobile robot that were created by the GA, which was the technique for generating an optimal path. They obtained rms errors of 0.0171 m/s for the forward velocity v_x, 0.0155 m/s for the lateral velocity v_y, and 0.0156 rad/s for the yaw angular velocity w. A. Medina-Santiago et al. [11] presented a multi-layer perceptron (MLP) with a BP algorithm to control the movements of a mobile robot that must generate a path and navigate avoiding obstacles using ultrasonic sensors. An Arduino embedded platform was used to implement the neural control, and the adaptability of the robot was tested with a group of people that acted as the dynamic obstacles. The authors concluded that NNs are great tools applicable in mobile robots evading obstacles since they can work with imprecise information. M. Duguleana and G. Mogan [9] used Q-learning (RL algorithm) and a feedback neural network to generate a collision-free trajectory for an autonomous robotic platform (PowerBot) within an uncertain workspace containing stationary and mobile entities. The network received as input the initial position of the mobile robot, the time sample, and the matrix of Q-values, while the output was a three-element vector that held the Cartesian values and the time. The weights of the neural network were updated after each step, using the adapt function, which received as an input the coordinates of the goal. The authors suggested that to minimize computation time the algorithm implementing the trajectory planner can be run in an embedded system. J. Yuan et al. [1] presented a dynamic path planning method based on a gated recurrent unit-recurrent neural network model for the problem of the path planning of a mobile robot in an unknown environment where inputs were derived from sample sets generated by an APF and an ant colony optimization algorithm (ACO). The input of the network was a dataset obtained by the laser sensor collecting surrounding environmental information, while the output was the velocity and the angle of the mobile robot obtained from the ACO algorithm and APF. In addition, the learning model was used to predict the control output angle at the next instant. The authors demonstrated with simulation and physical experiments that the network model could generate a reasonable trajectory in an unknown environment.

On the other hand, the training network needed samples generated by the teacher system, and sometimes it was impossible to reach the target point accurately.

Solutions involving LSTM neural networks have recently been presented. M. Inoue et al. [12] proposed a novel robot path planning method that combines the Rapidly Exploring Random Tree (RRT) and LSTM network where numerous paths are generated in the robot configuration space by the RRT method, and then a convolutional autoencoder and LSTM combination network is trained by them. The path generation ability of the trained network was investigated against the starting and goal points in an unknown environment with different static obstacle arrangements. The simulation results confirmed that the proposed network achieves high learning and generalization abilities. The authors S. Yin and A. Yuschenko [13] developed a method that can complete motion planning for a service mobile robot in one step using a convolutional LSTM network. The input of the network is a GRB picture with obstacles, target position, and starting position, while the output of the network is the linear and angular velocity of the robot. The convolution layer of the network serves to mark obstacles, target position and starting position; the LSTM layer describes the temporal characteristics of the movement; and the fully connected layer is used to smoothly adjust the linear and angular speed of the service mobile robot. The proposed method showed a good fault tolerance and was able to complete motion planning tasks in real time. The authors concluded that the LSTM layer is a fundamental part of the algorithm since the speed command given to the robot should have time characteristics, which means that the current speed command and the previous speed command must have a certain correlation to ensure continuity and smoothness. As for future work, they assume that their method can be applied for more complicated tasks, such as in the case of moving obstacles.

This paper proposes an LSTM neural network to achieve safe trajectory planning and navigation for a mobile robot from an initial point to an end point while avoiding a dynamic obstacle. The training examples were obtained by operating the mobile robot in a known dynamic environment and recording and labeling its sensor readings, target desired location, mobile robot location with respect to the odometry reference frame, and the velocity readings. The main contributions of this work can be summarized as follows: an LSTM network with an Adam optimization method is proposed to learn the trajectory planning strategy for the mobile robot in a dynamic environment; the algorithm is verified in a computationally simulated environment and then uploaded to GitHub, which is an open-source repository for code and software applications.

The results show an accuracy of training of above 99%. With shorter trajectories (< 2 m) the model can generalize other trajectories for which it was not specifically trained; however, it must be within a 15 cm radius of a trained destination position. In the case of longer and more complex trajectories, the mobile robot can successfully arrive at the destination point for which the model was trained. The LSTM network with the Adam optimization method lets a robot find a trajectory from the specified start location to the desired location while avoiding dynamic and static obstacles.

The rest of the paper is organized as follows: Section 2 theoretically describes the basis of artificial neural networks, recurrent neural networks, LSTM neural networks, Adam's optimization model, the robotic operating system framework, and the Gazebo simulator; Section 3 refers to the design and methodology of the experiments; Section 4 presents the results; and Section 5 exposes the conclusions and future work.

2. Materials and Methods

2.1. Artificial Neural Networks

ANNs are generalized mathematical models of biological nervous systems. The effects of the neuron synapses are represented mathematically by the connection weights that modulate the effect of the input signals, then a transfer function is used to depict the non-linear characteristic exhibited by neurons, and finally, to compute the neuron impulse it is necessary to consider the weighted sum of the input signals transformed by the transfer

function. Adjusting the weights of a neural network in concordance with the chosen learning algorithm will determine the learning capability of the ANN [14].

There are three classes of learning situations in neural networks: supervised, unsupervised, and reinforcement [14,15]. An ANN is usually used for supervised learning, a machine learning technique that allows the collection or production of data output after learning a classifier using the available training samples. In other words, supervised learning means that an external teacher provides the desired signals on individual output nodes. Some of the most famous examples include the BP algorithm, the delta rule, and the perceptron rule. In unsupervised learning an output unit is trained to respond to pattern groups within the input, and the system must develop its own representation of the input stimuli. Finally, in reinforcement learning the system will discover the actions that yield more rewards by trying them. The characteristics of the trial-and-error search and delayed reward represent the most important features of the reinforcement learning technique.

For the architecture of an ANN there are three types of neuron layers: input, hidden, and output. The behavior of the network will depend on the interaction between different neurons. For example, in feed-forward networks the signal flow goes from input to output units in a feed-forward direction, and the data processing can cover multiple layers of units, but no feedback connections exist [14]. They are also known as multi-layer perceptron (MLP) neural networks and are the most popular neural networks where input–output connections can be achieved by adjusting the associated weights in the network [16]. Feed-forward neural networks form the basis of many important neural networks currently used, such as convolutional neural networks (CNN) and recurrent neural networks (RNN). A. Yilmaz et al. [17] show an example of a feed-forward neural network in Figure 1.

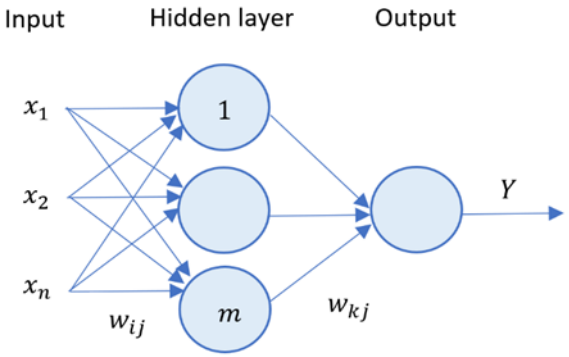


Figure 1. Architecture of a typical multilayered feed forward NN, where x_i is the neuron input, w_{ij} and w_{kj} are the weights, m represents the number of neurons in the hidden layer, and Y is the output value.

2.2. Recurrent Neural Networks

RNNs are widely used to analyze the structure of time series data [18]. They hold feedback connections. Contrary to feed-forward networks, the dynamical properties are crucial. In other words, some of the outputs are routed back as inputs with a delay, and these feedback neurons are the ones that keep the state variables (Figure 2).

The output of each neuron in the network is represented by Equation (1):

$$y^l{}_i = f(\sum_{j=0}^{N_{l-1}} w^l{}_{ji} x^l{}_j) \quad (1)$$

where l constitutes the layer, i the neuron in the layer l, N_{l-1} the number of neurons in the layer $l-1$, $w^l{}_{ji}$ the neurons' weights, $x^l{}_j$ an input coming from the output of neuron j in the layer $l-1$ (the inputs come from the sensors in the first layer), and $f()$ represents a nonlinear function [19].

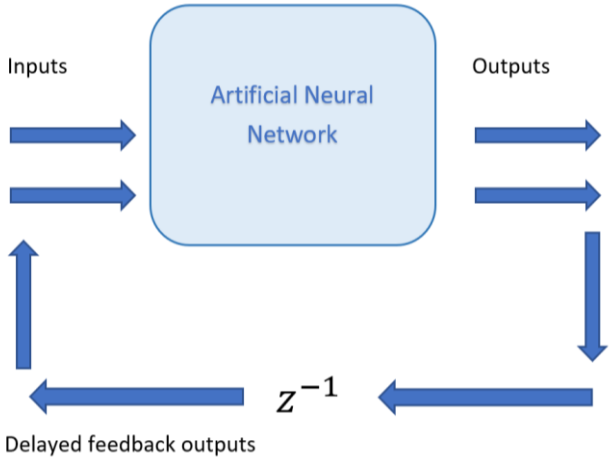


Figure 2. Recurrent neural network.

2.3. LSTM Neural Network

LSTM is a notable variant of an RNN that has been employed in many applications of sequence data. An LSTM has the advantage of having a continuous space memory which allows it to use the arbitrary length of past observations for sequence predictions [20].

Recently, the LSTM model has been applied to analyze vehicle trajectory sequences, as it effectively overcomes the vanishing gradient issue in naively designed RNNs [18]. A deep neural network (DNN) is an ANN with multiple layers between the input and output layers. In some cases, the neurons in the lower layers of a multi-layered network can hardly be updated or they even die, which blocks the DNNs from going deeper between the layers [21]. This is known as vanishing gradients, and they occur when the learning signals tend to zero with an increase in the number of layers in the DNN [22].

The LSTM network model consists of the cell memory that stores the summary of the past input sequence, and the gating mechanism by which the information flow between the input, output, and cell memory are controlled [23].

The following recursive equations describe how the LSTM works [18]:

$$f_t = \sigma(W_{uf}u_t + W_{hf}h_{t-1} + b_f). \tag{2}$$

$$i_t = \sigma(W_{ui}u_t + W_{hi}h_{t-1} + b_i) \tag{3}$$

$$o_t = \sigma(W_{uo}u_t + W_{ho}h_{t-1} + b_o). \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tan\mathrm{h}(W_{uc}u_t + W_{hc}h_{t-1} + b_c). \tag{5}$$

$$h_t = o_t \tan\mathrm{h} \odot (c_t). \tag{6}$$

where $\sigma(x) \triangleq \frac{1}{1+\exp(-x)}$ represents the sigmoid function (element wise), which is used as a powerful tool in the LSTM Model [23,24]; $x \odot y$ is the element wise product; u_t the input vector at current timestamp t; h_{t-1} is the output of the previous LSTM block (timestamp $t-1$); W_{ui}, W_{hi}, W_{uf}, W_{hf}, W_{uo}, W_{ho}, W_{uc}, and W_{hc} correspond to the linear transformation weight matrices; b_i, b_f, b_o, and b_c are the bias vectors; i_t, f_t, and o_t are the gating vectors; c_t is the cell memory state vector; and h_t is the state output vector. The gating vectors (2), (3), and (4) determine the amount of information for the cell memory to update, forget, and output its state. Then, the cell state and output are updated according to (5) and (6). The cell state can be reset or restored depending on the state of the forget gating vector. The two gating vectors i_t and o_t work in a similar way to regulate the input and output [18].

To sum up, the LSTM model is an ANN architecture that has managed to prove to overcome the vanishing gradient issue. It consists of three gates: the input, forget and output gates. These gates are the sigmoid activation function, meaning that they output a value between 0 and 1. The sigmoid function is used to obtain only positive output values

if certain features are needed to be kept or not, meaning that a value of 0 is going to be obtained if the gate blocks everything and a value of 1 is going to be obtained if the gate allows everything to pass through it. The forget gate (2) indicates which information will be thrown away, while (3) designates what new information is going to be stored in the cell state for the input gate. The output gate (4) provides the activation for the final output of the LSTM block at timestamp t (Figure 3) [25].

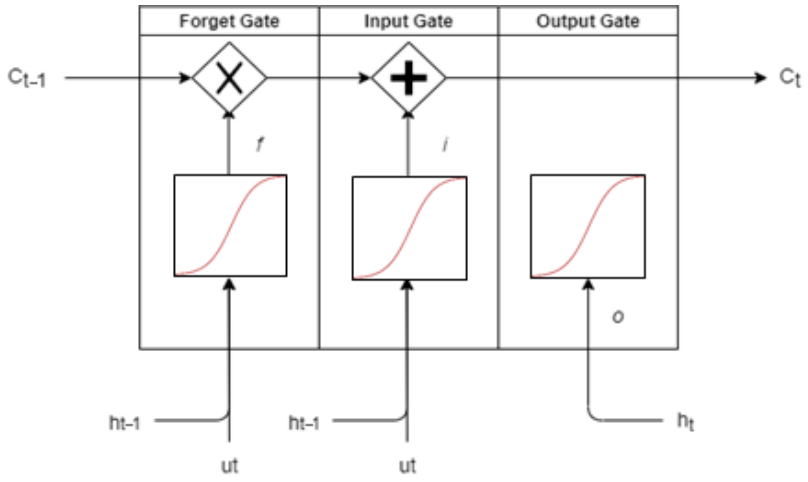


Figure 3. LSTM memory cell.

Lastly, the cell state is filtered and passed through the activation function which predicts what portion appears as the output of the current LSTM unit at timestamp t. The state output vector h_t passes from the current block through the SoftMax layer to get the predicted output y_t (Figure 4) [25].

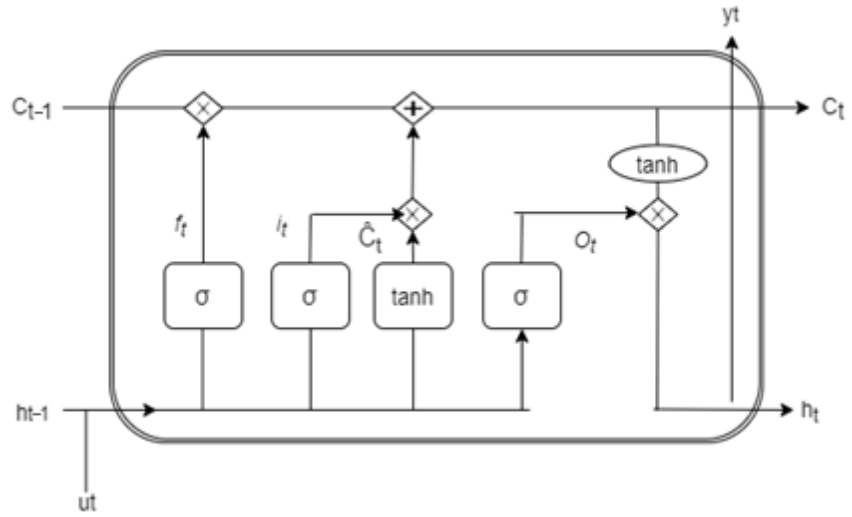


Figure 4. LSTM block at any timestamp.

It is common to append a SoftMax function as the final layer of a neural network as it converts the output real-valued scores to a normalized probability distribution that can be displayed to a user or used as input for other systems [26].

2.4. Adam Optimization Method

The Adam method is one of the most efficient stochastic optimization methods [27]. Some of the advantages of this method is that it is computationally efficient and invariant to diagonal rescaling of the gradients with little memory requirements [28]. The authors Z. Chang et al. [27] consider that the Adam optimization method can optimize the deep

learning model by finding a series of parameters to minimize the objective function. The main objective of the method is to find a set of parameters that minimize the mean squared error, which is a measure of the difference between the values predicted by a model and the observed values.

D. Kingma and J. Ba [29] state that the first-order gradient-based optimization method Adam, derived from adaptive moment estimation, is straightforward to implement, is well suited for problems that are large in terms of data and/or parameters and is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. Their results demonstrate that the method works well in practice and compares favorably to other stochastic optimization methods.

2.5. Robot Operating System (ROS)

ROS is a framework that offers a core set of software for operating robots that can be broadened by creating or using existing packages. There are thousands of packages available per stable distribution encapsulating algorithms and sensor drivers, among others. ROS programs can be created mainly in two programming languages: Python and C++. The main advantage of ROS is that it is a free and open source, making it possible to write robotic software that can be reused on distinct hardware platforms. Also, ROS is available for a wide variety of robots, such as mobile robots, manipulators, humanoid robots, autonomous vehicles, etc. ROS provides a peer-to-peer architecture that enables 'master' or 'slaves' components to dialogue directly with each other, synchronously or asynchronously. Finally, ROS is easy to use because its drivers or algorithms are contained in standalone executables, keeping its size down [30].

Gazebo

Gazebo is a multi-robot simulation tool that can provide accurate simulation for robots, sensors, and objects in 3D. It generates realistic sensor feedback and has a robust physics engine to produce interactions between objects, robots, and environments. Gazebo is also offered freely as a stand-alone software but has also been packaged along with ROS as a simulation tool [31]. The main advantage of Gazebo is that it offers high-quality graphics, and suitable programmatic and graphical user interfaces [32].

3. Experimental Setup

The following assumptions were considered before setting up the simulated environment:

1. The mobile robot operates in a workspace with several static obstacles and one dynamic obstacle.
2. The working environment of the mobile robot is a two-dimensional $x \times y$ space where x and y indicate coordinates in meters $x \times y$ space where x and y indicate coordinates in meters.
3. All dynamic parameters are known at each time instant, including the initial and final Cartesian position of the mobile robot, its location with respect to the odometry reference frame, and the velocity of the dynamic obstacle.
4. The robot is non-holonomic and is performing the planning in a 2D environment.
5. The dynamic obstacle is represented by another mobile robot.

3.1. Simulated Environment Setup

The ROS platform, its packages and dependencies, and the Gazebo simulation tool were completely installed using the Ubuntu 18.04.5 packages, which is a free and open-source software operating system, and a Debian-based Linux distribution. Both were synchronized so that the logic could be executed, and the mobile robot could act in its environment. The necessary additional software, such as Windows 10 and Microsoft Office were licensed by Tecnológico de Monterrey. The main virtual tools that were used to perform the experimentation are listed in Table 1 [33–36].

Table 1. Main tools for experimentation.

Tool	Version/Branch	Main Tasks
ROS Framework	Melodic	Hardware abstraction, low-level device control, implementation of commonly used functionality, message passing between processes, and package management.
Gazebo Simulator	9.0.0	Multi-robot and environment 3D simulation. Information acquisition and modification of the simulated world.
RVIZ Graphical Interface	1.13.13	Information visualization: sensor data, robot models, environment maps.
Python Programming Language	2.7.17	Interpreted, object-oriented, high level and flexible programming.
Atom Text-editor	1.53.0	Intuitive graphical user interface and cross-platform text-editor programming.

The mobile robot and its environment were simulated in Gazebo. The grid environment scenario was of 5 × 5 m surrounded by walls and six static obstacles. Subsequently, the multiple static obstacles were simulated as cylinders and placed in random occupancy grids.

The testing robot platform chosen for the simulation experiment was the TurtleBot 3 Waffle Pi by Robotics. A simulated accurate representation of the robot was launched in Gazebo inside the following scenarios shown in Figures 5–7. The URDF files containing the mechanical properties of the mobile robot were not modified to maintain the mobile robot's physical characteristics.

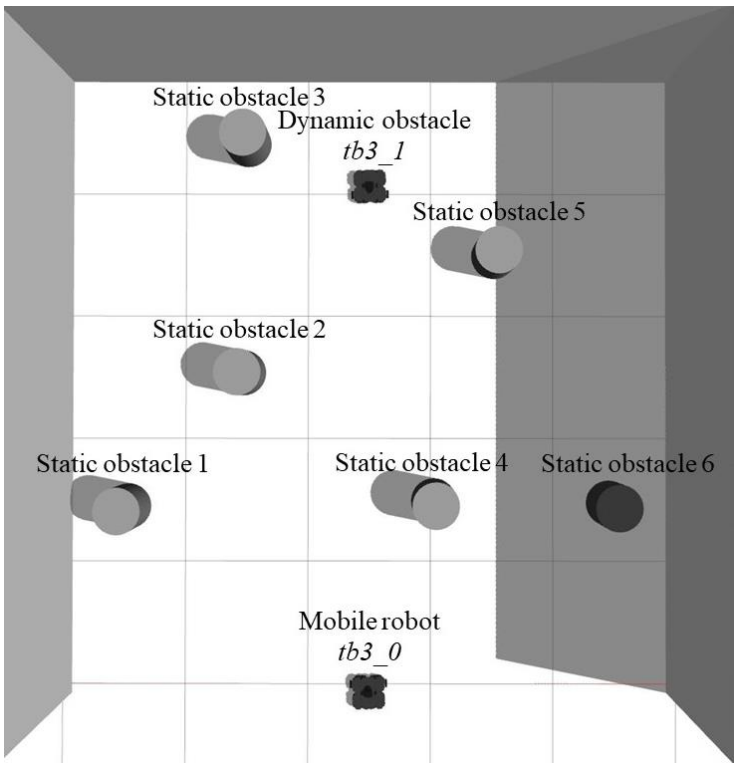


Figure 5. Simulated environment scenario 1 with labels.

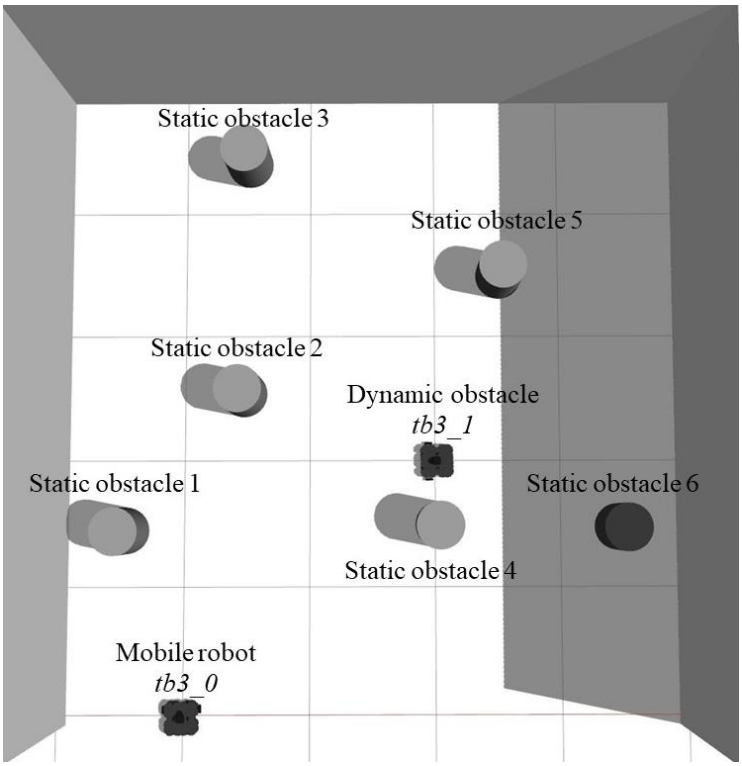


Figure 6. Simulated environment scenario 2 with labels.

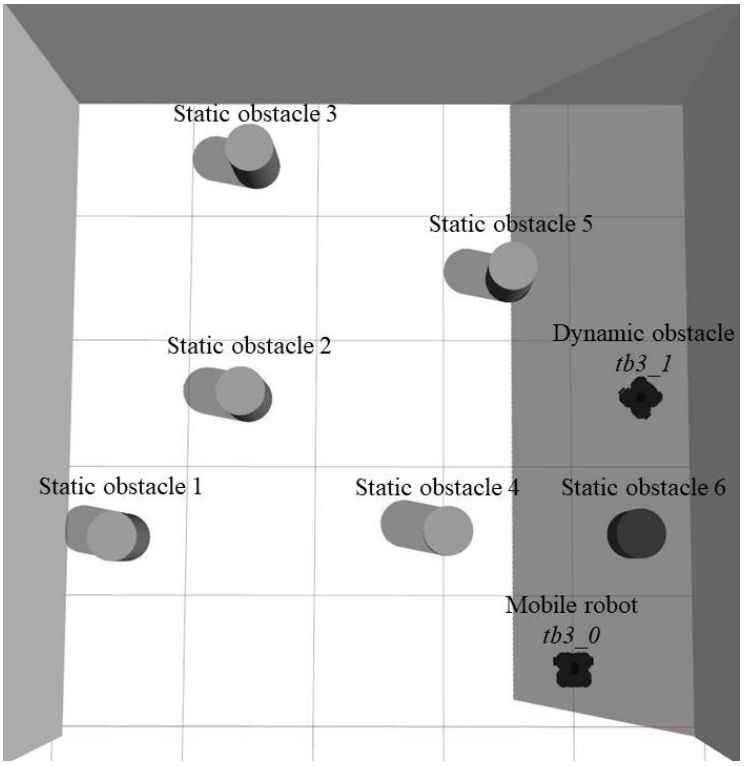


Figure 7. Simulated environment scenario 3 with labels.

As the dynamic obstacles can be any moving objects and/or persons, another TurtleBot 3 Waffle Pi was added to the simulation representing the dynamic obstacle, labeled as *tb*3_1, while the main mobile robot was marked as *tb*3_0. Three different scenarios, shown in Figures 5–7, were constructed to test the robustness of the proposed LSTM model

under different conditions for the dynamic obstacle and the mobile robot. Each simulation scenario could be called through the roslaunch tool, which is the standard method for starting multiple ROS nodes, placing multiple robots, and bringing up the simulated world in Gazebo [34].

The three scenarios are different for the starting position of the mobile robot and the dynamic obstacle; however, the position of the static obstacles remains fixed and is the same for all scenarios.

3.2. Sensor Data Acquisition

The physical and simulated TurtleBot 3 Waffle Pi mobile robot comes with two types of sensors: a 2D LiDAR LDS-01 sensor and an 8Mp V2 Raspberry Pi Camera [37]. The 2D LiDAR sensor is a 360 degrees laser distance sensor that has been widely used to perceive the environment. This type of sensor evaluates the surroundings in the form of a point cloud by measuring the intensity and the flight time of the laser reflected from the object after emitting it from the sensor [38]. The development of microcomputers and single board computers has helped to develop low-cost solutions for the domain of the mobile robots [39]. The Raspberry Pi is an example of this, and it allows an online environment for viewing through the Pi Camera.

The acquisition sensor data that enters the LSTM neural network comes from the LiDAR LDS-01, which has an operating range from 12 cm to 350 cm, a sampling rate of 1.8 kHz and an angular resolution of 1° [37].

The authors F. Shamsfakhr and B. Sadeghibigham [40] state that it is necessary to apply an appropriate dimensional reduction method to a dataset and extract the most meaningful principal features of the data that cover a reasonable percentage of the explained variance of the dataset, in order to avoid a slow convergence and learning process of a neural network due to the large number of features of the training patterns. For this reason, only values obtained every two degrees within a laser scan sample from 0 to 180° at the front of the robot were considered. A ROS node named "get_values" was initialized to subscribe to the laser scan topic of the TurtleBot3 (mobile robot) and obtain the raw LiDAR scan data. A function was defined to achieve an appropriate dimensionality data reduction. For every time instance (scan sample), a list with 90 features describing the distances from the reference frame (LiDAR) to an obstacle (static or dynamic) was obtained.

3.3. Dynamic Obstacle Trajectory

To move the dynamic obstacle, which was another TurtleBot3 Waffle Pi, a program was coded setting a constant velocity with different initial and final configuration points. For example, for the simulated environment scenario 1, Figure 5, the dynamic obstacle $tb3\_1$ was placed initially in $(2.5, 4.0)$ and was moving vertically towards the mobile robot $tb3\_0$ with a constant speed of 0.2 m/s until it reached the point $(2.5, 1.0)$. For the simulated environment scenario 2, Figure 6, the dynamic obstacle $tb3\_1$ was placed initially in $(3.0, 2.0)$ and was moving horizontally towards the mobile robot $tb3\_0$ with a constant speed of 0.2 m/s until it reached the point $(0.5, 2.0)$. Finally, for the simulated environment scenario 3, Figure 7, the dynamic obstacle $tb3\_1$ was placed initially in $(4.5, 2.5)$ and was moving diagonally towards the mobile robot $tb3\_0$ with a constant speed of 0.2 m/s until it reached the point $(2.5, 0.5)$. A ROS node, "speed_controller", was initialized to subscribe to the odometry topic of the dynamic obstacle to obtain its position, and to publish to the twist (velocity) topic.

3.4. LSTM Neural Network

3.4.1. Training Data

To train the network, the mobile robot was operated in the proposed simulated dynamic environments and the velocities and sensor data labels were recorded at each time instance, which in this case, was every 0.1 s. Two additional parameters were added: the desired target point location in $(x,\ y)$ for the mobile robot $tb3\_0$ and its odometry pose

in $(x,\ y)$ coming from the odometry topic of the ROS. The target point was fixed, while the odometry pose of the robot varied every time instance $(0.1\ \mathrm{s})$.

The trajectory planning and navigation examples were provided by the user to the network to be trained using the teleop command, which consisted of operating the mobile robot in the terminal window with specific keyboard keys that control the linear velocity in x and the angular velocity in z of the mobile robot $tb3\_0$.

The data was stored in a .csv file, where all the columns represent the 90 features coming from the LiDAR, and the target point and odometry pose location, while the rows represent each laser scanned packed with those features at each timestamp. The last two columns included are the linear velocity in x and the angular velocity in z recorded at each timestamp. It is crucial to mention that if a value is marked as infinite, it means that it was further away from the LiDAR reference frame, which had an operating range from 12 cm to 350 cm, so those values were replaced by the maximum range value of the sensor. That way it could be guaranteed that the proposed network would converge at some point. The roslaunch tool was implemented again to start two ROS nodes: the one with the obstacle controller algorithm, "speed_controller", and the one with the training data and .csv storage program, "get_values".

3.4.2. LSTM Model

The proposed Adam optimization algorithm for LSTM architecture consisted of an input layer of 188 neurons containing the 94 features describing the laser scan data (distance to obstacles), the target location $(x,\ y)$, the mobile robot odometry location $(x,\ y)$ from timestamp t and the same 94 features from timestamp $t-1$, then three hidden layers of 90 neurons each, and an output layer of 2 neurons that returned the linear velocity in x and angular velocity in z of the mobile robot $tb3\_0$.

The loss of the network was calculated as the mean squared error, optimized with the Adam model, and trained with 500 epochs and a batch size of 32. The recorded data in .csv format was reshaped to fit the LSTM layers *(batch size, time steps, features)*, considering the features and the previous time step. Figure 8 describes the LSTM model architecture, where $(T_{Px},\ T_{Py})$ represents the desired target location; $(C_{Px},\ C_{Py})$ is the odometry pose of $tb3\_0$ at timestamp t; $d_1 - d_{90}$ are the 90 laser scan features at timestamp t and $t-1$.

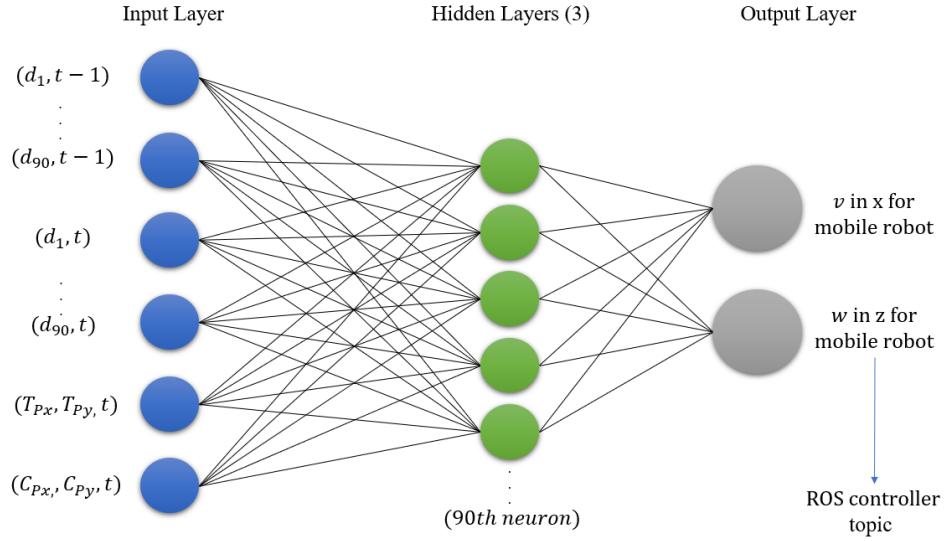


Figure 8. LSTM model architecture.

The number of hidden neurons was empirically found to be convergent of error to a minimum threshold error. The number of hidden layers was also found empirically, as with one hidden layer it is difficult to learn the network within a specified error limit [40]. The proposed LSTM model deals with 461,542 trainable parameters. Overfitting is a

serious problem in DNNs with many parameters for training. Dropout is the solution technique for overfitting, since it randomly drops units along with their connections from the neural network during training, preventing the units from excessive co-adaptation. That is why 20% dropout layers were added between the hidden layers [41]. A ROS node, "testing", was initialized to subscribe to the laser scan topic and obtain online LiDAR sensor data, and to publish to the twist (velocity) topic with the LSTM model predicted velocities of *tb*3_0. The roslaunch tool was used to start two ROS nodes: the one with the obstacle controller algorithm, "speed_controller", and the one with the LSTM model testing navigation, "testing", so that the navigation of the mobile robot with the proposed model could be tested in the simulated environment scenarios in Gazebo, Figures 5–7.

4. Results

The first experiments were implemented with one trajectory example for the mobile robot *tb*3_0. Figure 9 describes the velocity commands given by the user to the LSTM model. The linear velocity in x in m/s of the mobile robot is represented by the green line, while the angular velocity in z in rad/s is represented by the red line. The results showed that training for only one trajectory was not enough for the model to learn to reach different target locations while evading a dynamic obstacle.

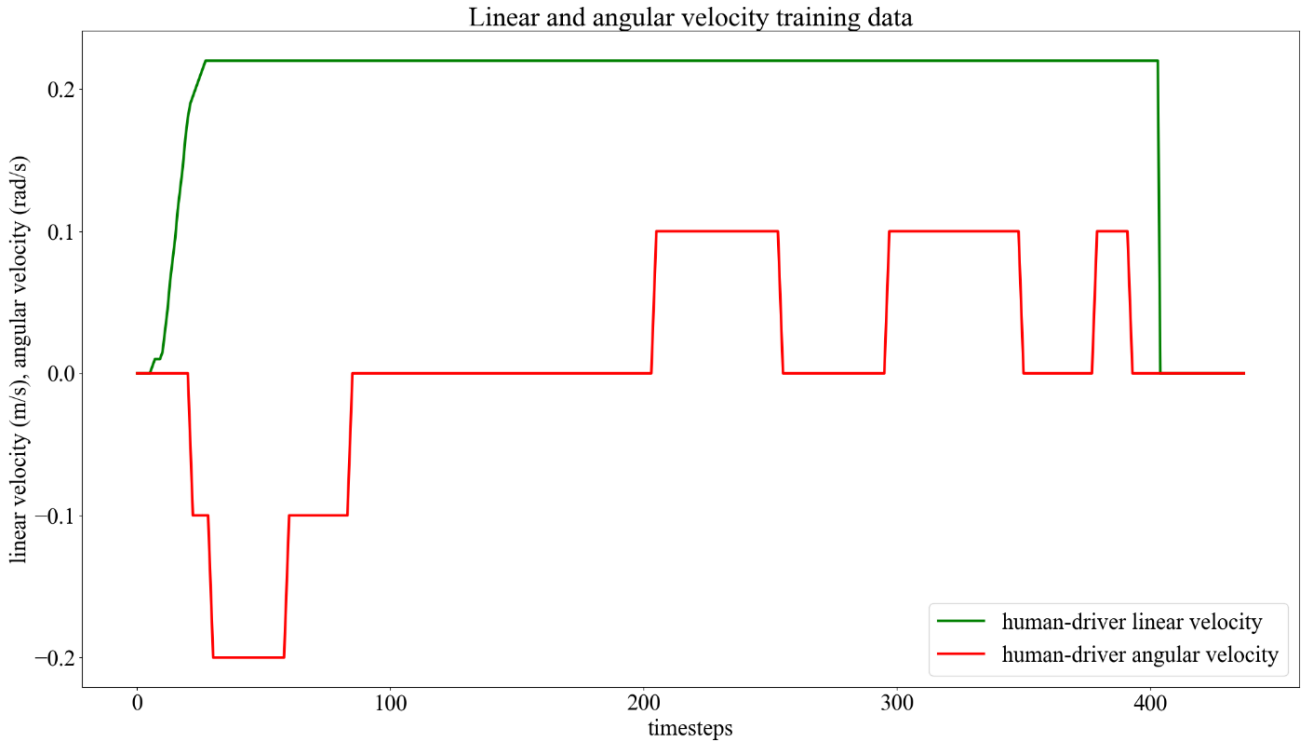


Figure 9. Velocity commands for one trajectory given by the user for training.

Further experimentation included several trajectories to train the network. Figure 10 describes many trajectories and navigation examples provided by the user through the velocity commands to the LSTM model. The linear velocity in x in m/s of the mobile robot is represented by the green line, while the angular velocity in z in rad/s is represented by the red line.

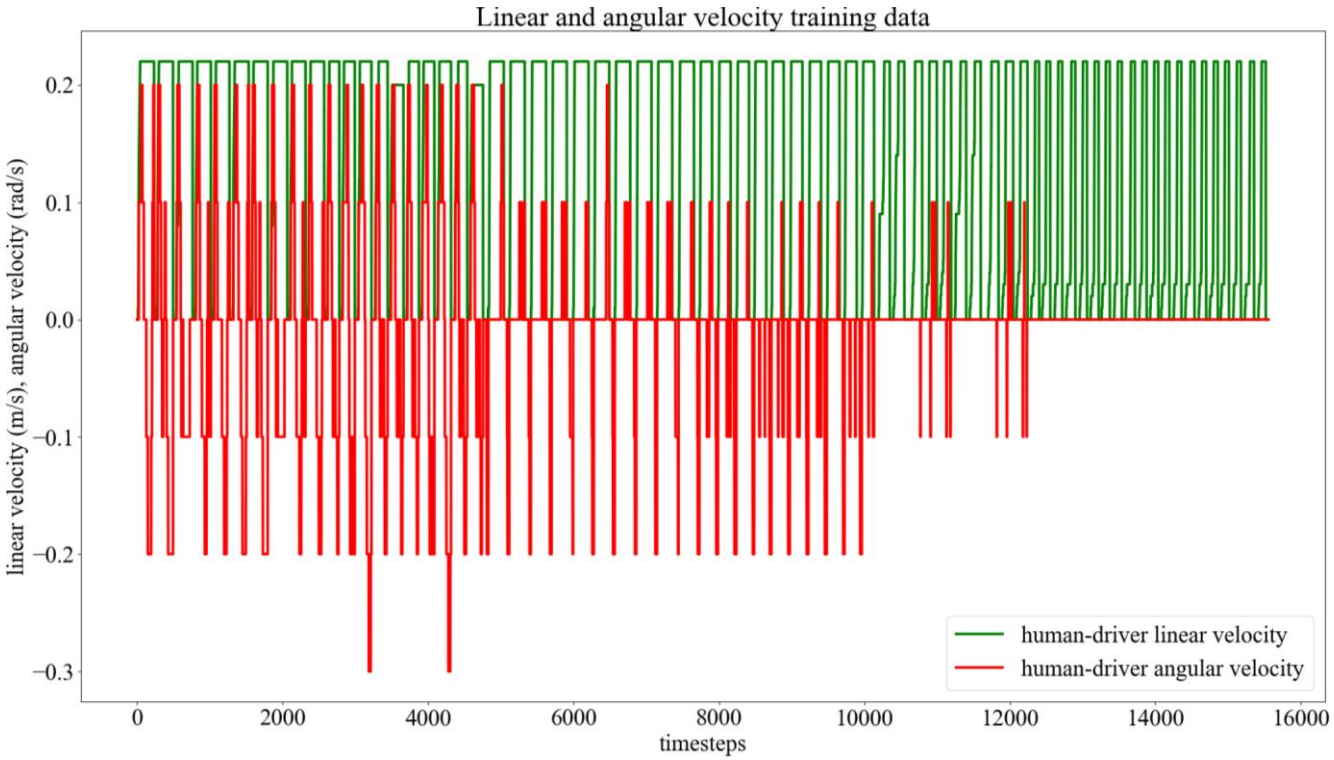


Figure 10. Velocity commands for several trajectories given by the user for training.

To train the network it was necessary to provide trajectory examples. Figure 9 shows only one trajectory example provided by the user to the network. The results show that one trajectory alone is not enough for the model to learn to reach different target locations while evading a dynamic obstacle. Therefore, more trajectories were needed, as shown in Figure 10, where 70 trajectories with different starting and target points were provided to the network. Each trajectory was completed within a given timestep number.

The trajectory in Figure 9 was completed with 400 timesteps. Each trajectory in Figure 10 has a different timestep length given the proximity of the target points, which is why Figure 10 has near to 16,000 timesteps.

After training the model with several trajectories, an MSE of 0.0002 m/s for the linear velocity in x, an MSE of 0.0004 rad/s for the angular velocity in z of $tb3\_0$, and an accuracy of 99.24% were obtained Figure 11. The loss curve in Figure 12 shows that the proposed LSTM model reached a consistent minimum MSE over the training runs/epochs.

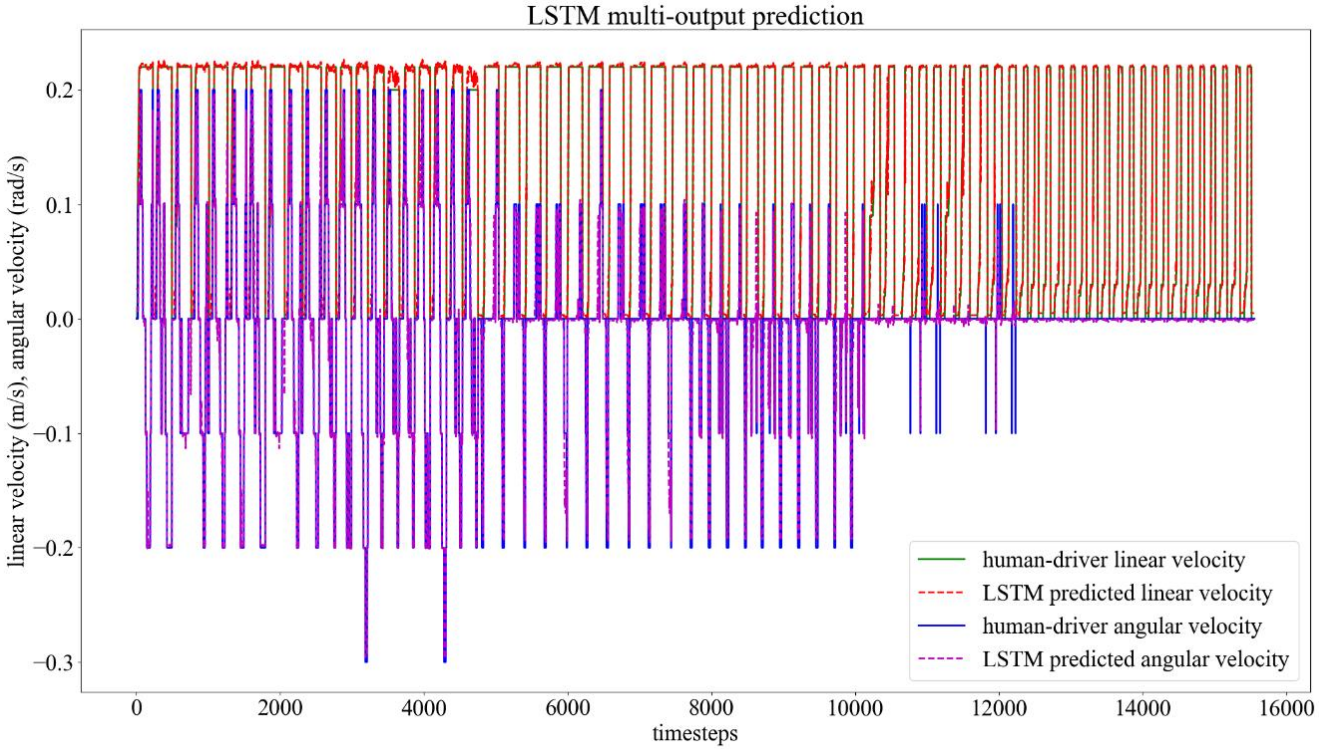


Figure 11. LSTM multi-output prediction for different trajectories at variable speeds.

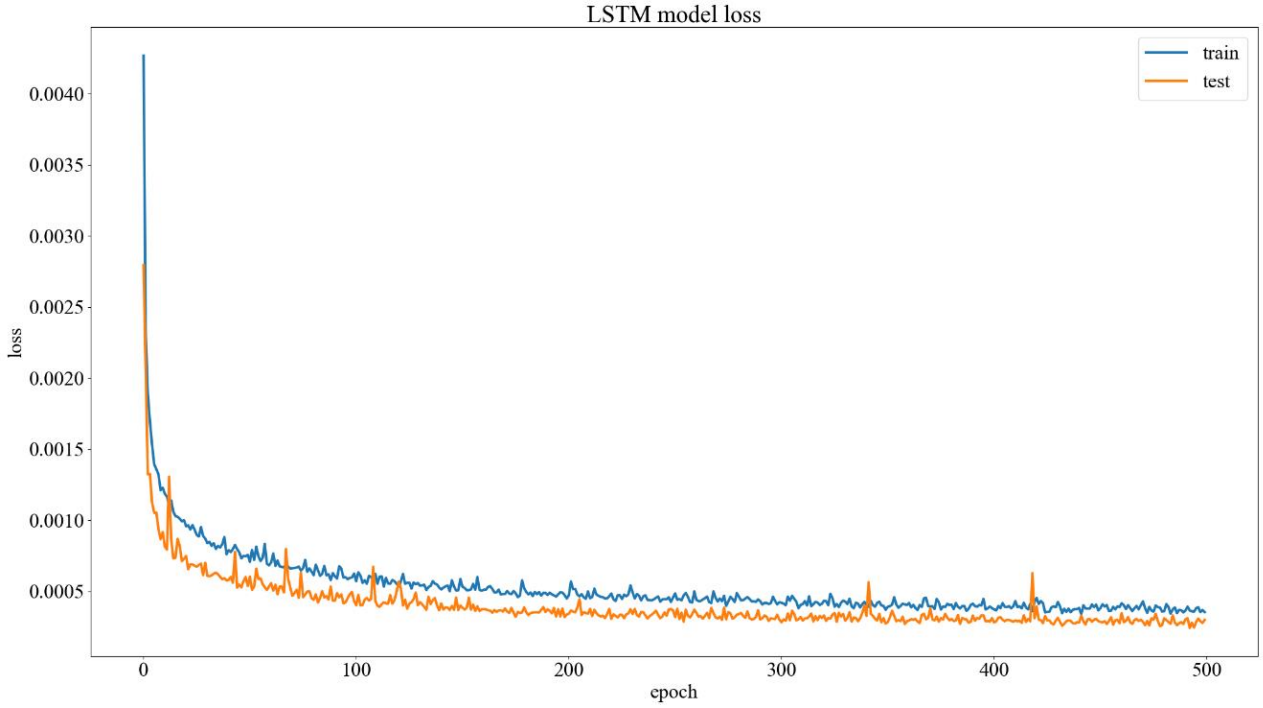


Figure 12. Model train loss vs. validation loss for many velocity commands for different trajectories.

The following images in Figures 13–15 show the simulation results in Gazebo, where the goal location is indicated by a green circle, and the path followed by the dynamic obstacle is marked manually by a red straight arrow.

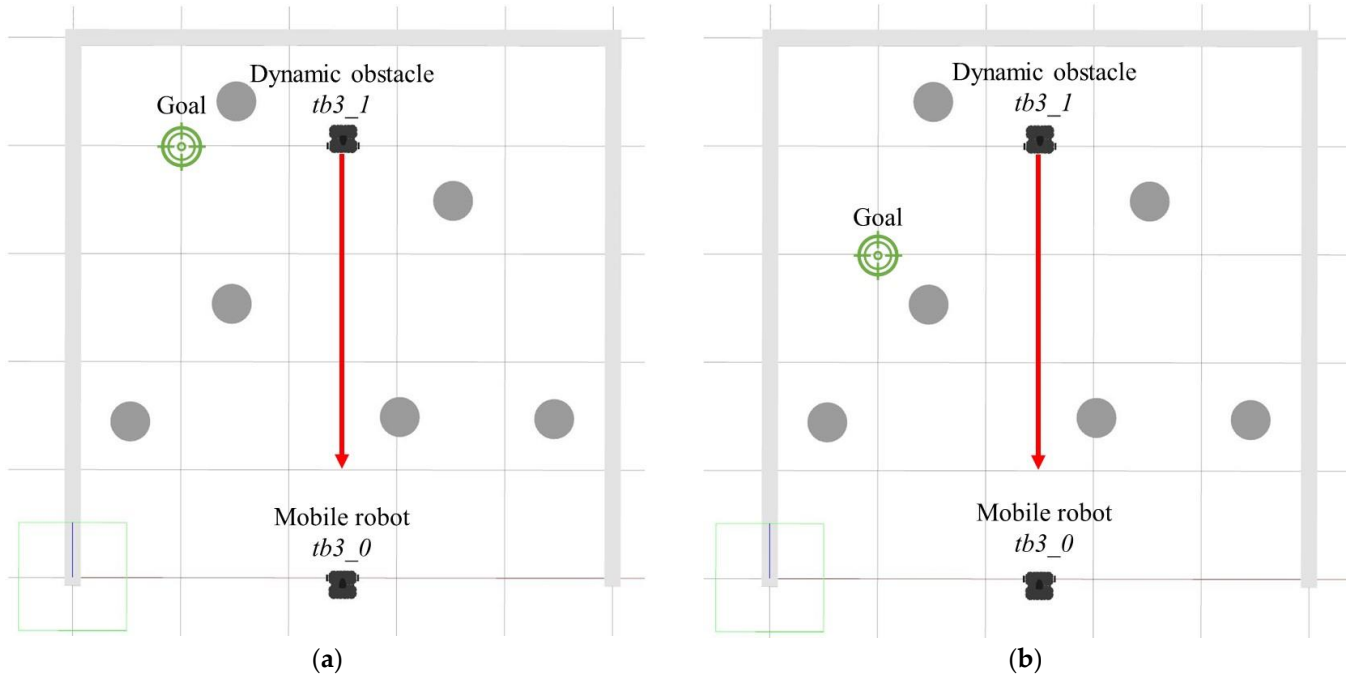


Figure 13. (**a**) Gazebo simulation for dynamic environment scenario 1 with goal point of $(1.0, 4.0)$ for *tb*3_0, (**b**) Gazebo simulation for dynamic environment scenario 1 with goal point of $(1.0, 3.0)$ for *tb*3_0.

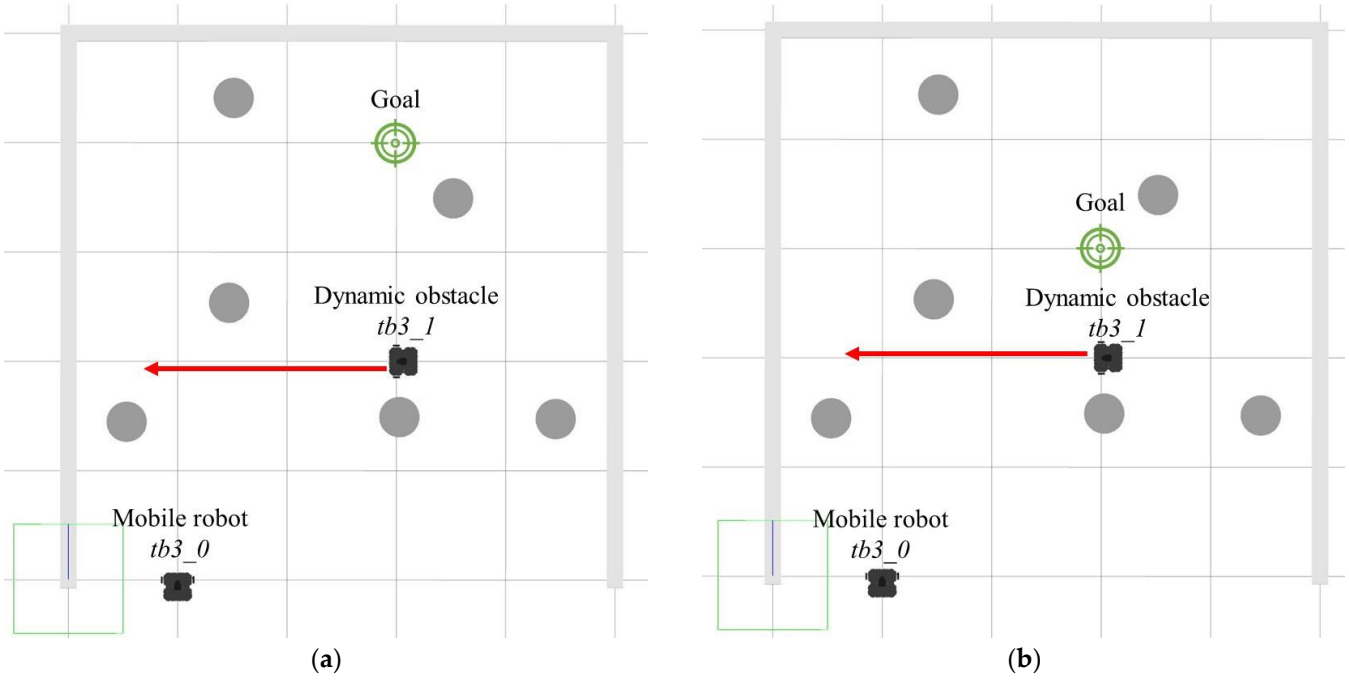


Figure 14. (**a**) Gazebo simulation for dynamic environment scenario 2 with goal point of $(3.0, 4.0)$ for *tb*3_0, (**b**) Gazebo simulation for dynamic environment scenario 2 with goal point of $(3.0, 3.0)$ for *tb*3_0.

In Rviz, the obstacles marked by green are the ones that the sensor sees at that time, considering they are within the range of the sensor, including the dynamic obstacle. Although all obstacles and objects always appear as visual elements in the simulation environment in Gazebo, in Rviz, the dynamic or static obstacles will not be seen if they are out of the sensor's field of view.

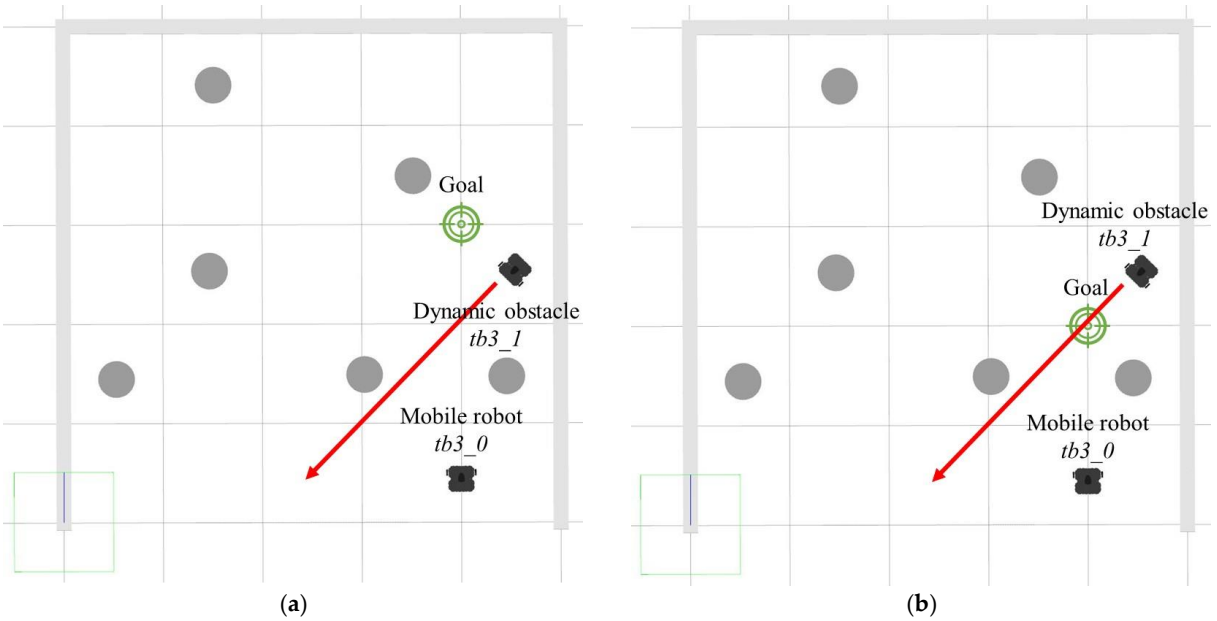


Figure 15. (**a**) Gazebo simulation for dynamic environment scenario 3 with goal point of $(4.0, 3.0)$ Figure 3. (**b**) Gazebo simulation for dynamic environment scenario 3 with goal point of $(4.0, 2.0)$ for *tb*3_0.

Figures 16–18 show, in Rviz, the mobile robot model *tb*3_0; the dynamic obstacle model *tb*3_1; the map of the 5×5 m simulated world for the experimentation; the path followed by *tb*3_0 in green; the path followed by *tb*3_1 in red; and the information captured in green from the *tb*3_0 LiDAR sensor, which has an operating range from 12 cm to 350 cm, a sampling rate of 1.8 kHz and an angular resolution of 1° [36]. The mobile robot *tb*3_0 arrives at the desired goal point evading the dynamic obstacle *tb*3_1 in each case scenario.

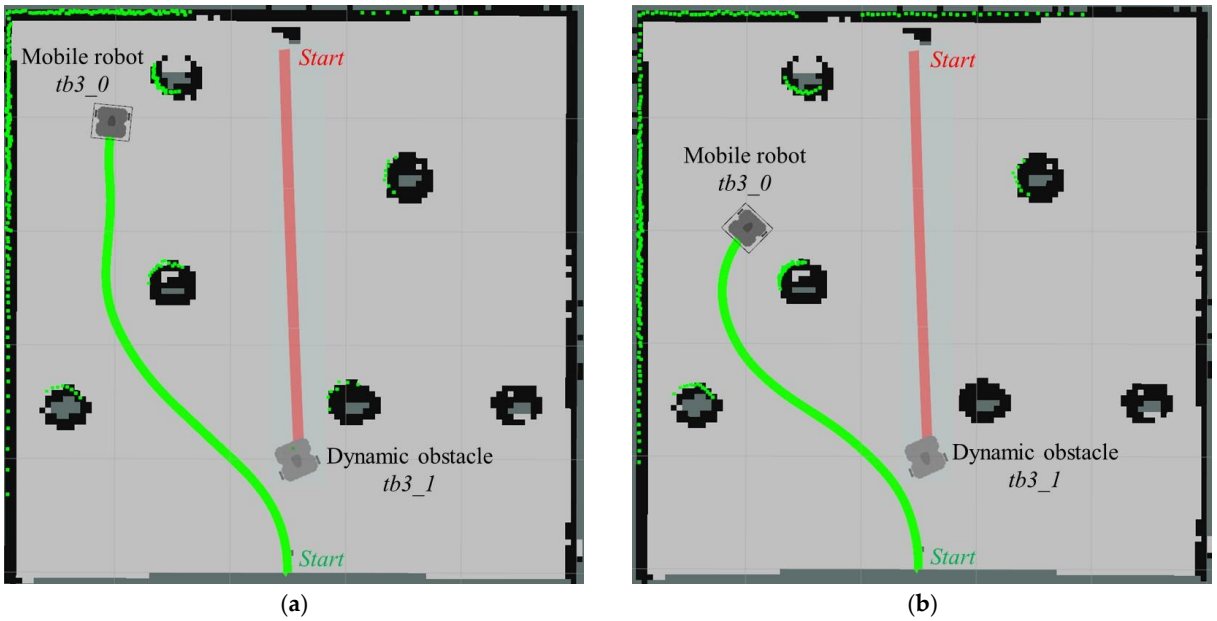
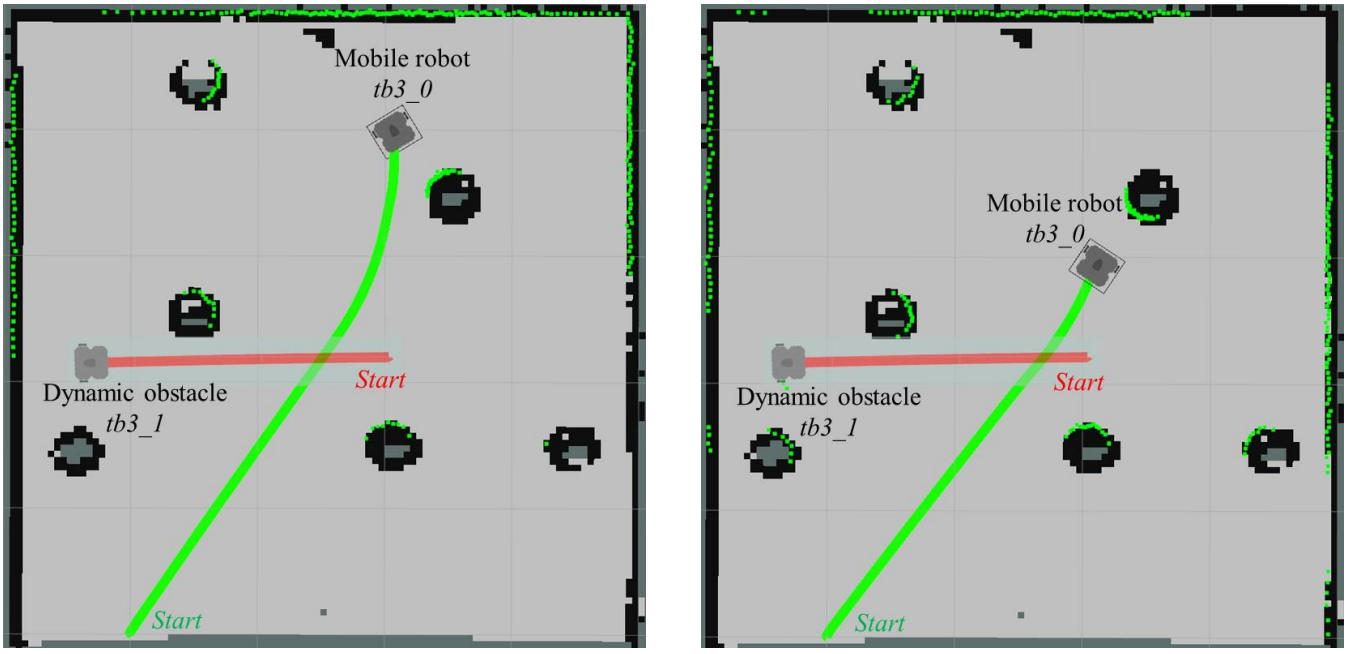
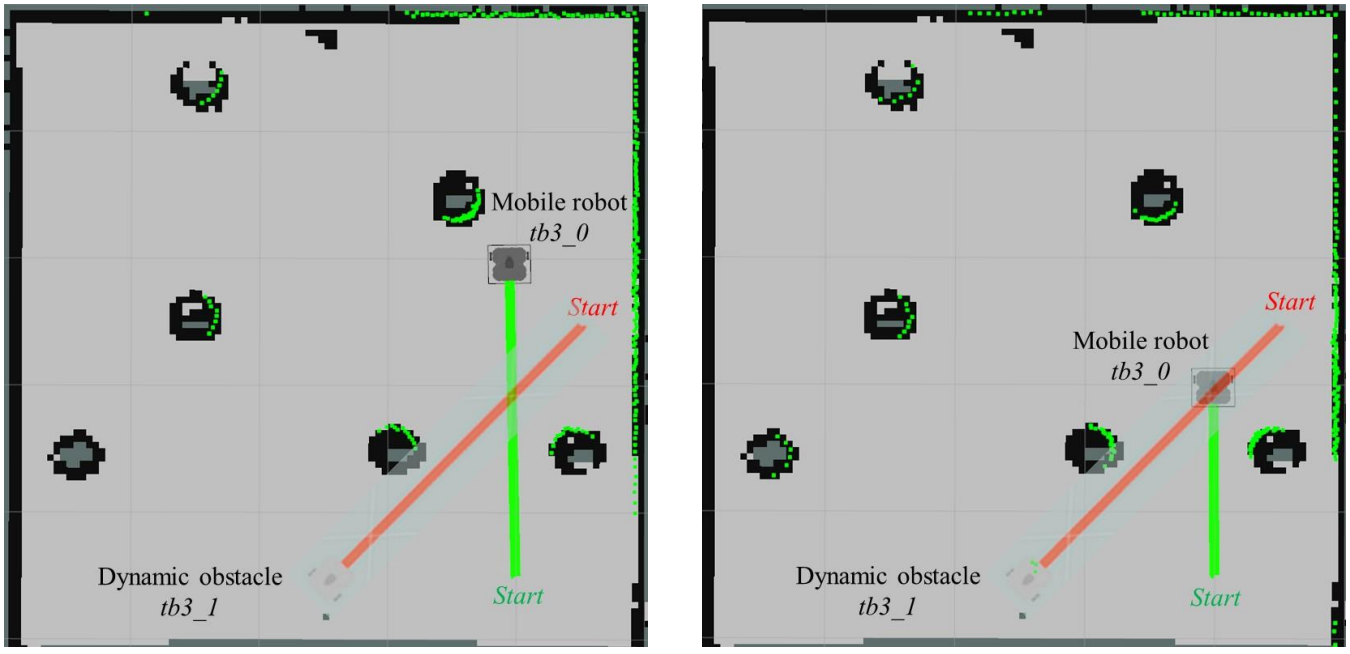


Figure 16. (**a**) Rviz simulation for dynamic environment scenario 1 after testing the LSTM model with goal point of $(1.0, 4.0)$ for *tb*3_0, (**b**) Rviz simulation for dynamic environment scenario 1 after testing the LSTM model with goal point of $(1.0, 3.0)$ for *tb*3_0.



(**a**) (**b**)

Figure 17. (**a**) Rviz simulation for dynamic environment scenario 2 after testing the LSTM model with goal point of $(3.0, 4.0)$ for *tb*3_0, (**b**) Rviz simulation for dynamic environment scenario 2 after testing LSTM model with goal point of $(3.0, 3.0)$ for *tb*3_0.



(**a**) (**b**)

Figure 18. (**a**) Rviz simulation for dynamic environment scenario 3 after testing the LSTM model with goal point of $(4.0, 3.0)$ for *tb*3_0, (**b**) Rviz simulation for dynamic environment scenario 3 after testing LSTM model with goal point of $(4.0, 2.0)$ for *tb*3_0.

For the simulated environment scenario 1 (Figure 16), the dynamic obstacle *tb*3_1 was placed initially at $(2.5, 4.0)$ and was moving vertically towards the mobile robot *tb*3_0 with a constant speed of 0.2 m/s until it reached the point $(2.5, 1.0)$. For the simulated environment scenario 2 (Figure 17), the dynamic obstacle *tb*3_1 was placed initially at $(3.0, 2.0)$ and was moving horizontally towards the mobile robot *tb*3_0 with a constant

speed of 0.2 m/s until it reached the point $(0.5, 2.0)$. Finally, for the simulated environment scenario 3 (Figure 18), the dynamic obstacle $tb3\_1$ was placed initially at $(4.5, 2.5)$ and was moving diagonally towards the mobile robot $tb3\_0$ with a constant speed of 0.2 m/s until it reached the point $(2.5, 0.5)$.

Figures 17 and 18 show an intersection of the green and red path lines of the mobile robot and the dynamic obstacle, but there was no collision, since $tb3\_0$ waited for $tb3\_1$ to pass.

Following the training of the LSTM model (with several examples of trajectories with six different target points, three distinct dynamic obstacles behaviors, and three unalike initial conditions for the mobile robot), an MSE of 0.0002 m/s for the linear velocity in x, an MSE of 0.0004 rad/s for the angular velocity in z of $tb3\_0$, rms errors of 0.0141 m/s for the linear velocity in x, 0.02 rad/s for the angular velocity in z, and an accuracy of 99.24 % were obtained. This means that the more training examples the neural network learns, the greater the precision and the easier it is for the model to be able to navigate to different target points while evading the dynamic obstacle.

The proposed LSTM algorithm was compared in terms of distance (m) and time (s) with the Dijkstra's algorithm, which is the default global planner in the ROS turtlebot3 packages. This type of algorithm is mainly used for determining the shortest paths between nodes in a graph that rely purely on local path cost [42].

D. Fox et al. [43] state that the collision avoidance approaches for a mobile robot can be divided into two categories: global and local. The local planner in the ROS packages is the Dynamic Window Approach (DWA), which is one of the most used methods for local obstacle avoidance in mobile robots. This method was executed with a fixed frequency, meaning that only a set of velocities could be commanded to the robot due to its acceleration and velocity limits, then a reward function was proposed to select the best velocities to execute [44]. Table 2 shows the comparison made between the two approaches (LSTM proposed model, Dijkstra/DWA model) in the environment scenarios 1, 2, and 3. Figures 5–7 with the same tested goal points of Figures 13–15 for each scenario.

Table 2. Comparison between LSTM and Dijkstra/DWA models.

Test Number	Time (s)		Distance Traveled by tb3_0 (m)	
	LSTM	**Dijkstra/DWA**	**LSTM**	**Dijkstra/DWA**
Test 1 Target Point (1.0,4.0)	22.86	24.04	4.6982	4.8196
Test 2 Target Point (1.0, 3.0)	20.41	18.01	4.0438	3.8037
Test 3 Target Point (3.0, 4.0)	28.10	30.15	4.5577	4.6788
Test 4 Target Point (3.0, 3.0)	22.39	22.72	3.6177	3.6329
Test 5 Target Point (4.0, 3.0)	16.95	Collision	2.6982	Collision
Test 6 Target Point (4.0, 2.0)	10.34	Collision	1.7993	Collision

In Tests 1 and 3, the LSTM solution had a better performance than the default solution for both time and distance wise. Meanwhile, in Test 2 the opposite occurs, where the training method slowed the robot with the LSTM. This happened because the mobile robot was trained to wait for the dynamic obstacle to pass. Test 4 was very similar in both solutions. Finally in the last two tests, the state-of-the-art solution crashed in narrow spaces while the proposed model showed its robustness and did not crash into the obstacles.

5. Conclusions

An environment with the mobile robot, and multiple static obstacles and a dynamic obstacle located in different positions within the same occupancy grid map was computationally simulated. LiDAR scan values obtained every two degrees within a laser scan sample from 0 to 180° at the front of the robot were considered. The training examples were obtained by operating the mobile robot in a known dynamic environment and recording and labeling its sensor readings, target desired location, mobile robot location with respect to its odometry reference frame, and the velocity readings. Finally, an Adam optimization method for the LSTM model was created and trained with several examples of trajectories composed of the velocity commands given by the user, with six different target points, three distinct dynamic obstacles behaviors, and three unalike initial conditions for the mobile robot. Results showed an accuracy of training of above 99%. An MSE of 0.0002 m/s for the linear velocity in x, and an MSE of 0.0004 rad/s for the angular velocity in z of $tb3\_0$ were obtained. In other words, an error of $\approx$ 0.1466% for the linear velocity in x was obtained, considering an average human-driver linear velocity of 0.1364 m/s, and an error of $\approx$ 4.9535% for the angular velocity in z was obtained, considering an average human-driver angular velocity of -0.0081 rad/s.

With shorter trajectories (< 2 m) the model can generalize other trajectories for which it was not specifically trained; however, it must be within a 15 cm radius of a trained destination position. In the case of longer and more complex trajectories the mobile robot can successfully arrive at the destination point for which the model was trained. The LSTM network with an Adam optimization method allows a robot to find a trajectory from the specified start location to the desired location while avoiding dynamic and static obstacles.

In the future, the method proposed may be applied for more complex tasks such as: planning trajectories for a group of mobile robots; testing by changing the velocity and path of the dynamic obstacle; adding more dynamic obstacles; and considering more training example trajectories composed of the velocity commands given by the user to strengthen the model. Furthermore, physical experiments with the TurtleBot 3 Waffle Pi could be executed to validate the model.

Author Contributions: Conceptualization, A.M.-L., A.G.-E. and J.A.E.C.; methodology, A.M.-L., A.G.-E. and J.A.E.C.; software, A.M.-L. and J.A.E.C.; validation, A.M.-L., A.G.-E., J.A.E.C., E.C.-U. and S.R.C.-R.; formal analysis, A.M.-L. and A.G.-E.; investigation, A.M.-L.; resources, J.A.E.C.; data curation, A.M.-L.; writing—original draft preparation, A.M.-L.; writing—review and editing, A.M.-L., A.G.-E., J.A.E.C., E.C.-U. and S.R.C.-R.; visualization, A.M.-L.; supervision, A.G.-E. and J.A.E.C.; project administration, A.M.-L. and A.G.-E.; funding acquisition, J.A.E.C. 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: Not applicable.

Acknowledgments: Authors would like to acknowledge the support of Tecnológico de Monterrey, and the financial support from CONACyT for MSc studies of the first author.

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

References

1. Yuan, J.; Wang, H.; Lin, C.; Liu, D.; Yu, D. A Novel GRU-RNN Network Model for Dynamic Path Planning of Mobile Robot. *IEEE Access* **2019**, *7*, 15140–15151. [CrossRef]
2. Europeana, P.E. The Use of Artificial Intelligence in Autonomous Mobile Robots. 1999. Available online: https://www.researchgate.net/publication/2379199 (accessed on 22 March 2020).
3. Fang, W.; Chao, F.; Yang, L.; Lin, C.M.; Shang, C.; Zhou, C.; Shen, Q. A recurrent emotional CMAC neural network controller for vision-based mobile robots. *Neurocomputing* **2019**, *334*, 227–238. [CrossRef]

4. Maw, A.A.; Tyan, M.; Nguyen, T.A.; Lee, J.W. iADA*-RL: Anytime Graph-Based Path Planning with Deep Reinforcement Learning for an Autonomous UAV. *Appl. Sci.* **2021**, *11*, 3948. [CrossRef]
5. Al-Taharwa, I.; Sheta, A.; Al-Weshah, M. A Mobile Robot Path Planning Using Genetic Algorithm in Static Environment. *J. Comput. Sci.* **2008**, *4*, 341–344. [CrossRef]
6. Bakdi, A.; Hentout, A.; Boutami, H.; Maoudj, A.; Hachour, O.; Bouzouia, B. Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control. *Robot. Auton. Syst.* **2017**, *89*, 95–109. [CrossRef]
7. Orozco-Rosas, U.; Montiel, O.; Sepúlveda, R. Mobile robot path planning using membrane evolutionary artificial potential field. *Appl. Soft Comput. J.* **2019**, *77*, 236–251. [CrossRef]
8. Low, E.S.; Ong, P.; Cheah, K.C. Solving the optimal path planning of a mobile robot using improved Q-learning, Rob. *Auton. Syst.* **2019**, *115*, 143–161. [CrossRef]
9. Duguleana, M.; Mogan, G. Neural networks based reinforcement learning for mobile robots obstacle avoidance. *Expert Syst. Appl.* **2016**, *62*, 104–115. [CrossRef]
10. Noguchi, N.; Terao, H. Path planning of an agricultural mobile robot by neural network and genetic algorithm. *Comput. Electron. Agric.* **1997**, *18*, 187–204. [CrossRef]
11. Medina-Santiago, A.; Camas-Anzueto, J.L.; Vazquez-Feijoo, J.A.; Hernández-de León, H.R.; Mota-Grajales, R. Neural control system in obstacle avoidance in mobile robots using ultrasonic sensors. *J. Appl. Res. Technol.* **2014**, *12*, 104–110. [CrossRef]
12. Inoue, M.; Yamashita, T.; Nishida, T. Robot path planning by LSTM network under changing environment. In *Advances in Computer Communication and Computational Sciences*; Springer: Singapore, 2019; Volume 759. [CrossRef]
13. Yin, S.; Yuschenko, A. Planning of service mobile robot based on convolutional LSTM network. *J. Phys. Conf. Ser.* **2021**, *1828*, 012002. [CrossRef]
14. Abraham, A. Nature and Scope of AI Techniques. In *Handbook of Measuring System Design*; Oklahoma State University: Stillwater, OK, USA, 2005. [CrossRef]
15. Adam, B.; Smith, I.F.C. Reinforcement Learning for Structural Control. 2008. Available online: http://cedb.asce.orgcopyrightasce (accessed on 23 March 2020).
16. Minemoto, T.; Isokawa, T.; Nishimura, H.; Matsui, N. Feed forward neural network with random quaternionic neurons. *Signal Process.* **2017**, *136*, 59–68. [CrossRef]
17. Yılmaz, A.C.; Aci, C.I.; Aydin, K. MFFNN and GRNN Models for Prediction of Energy Equivalent Speed Values of Involvements in Traffic Accidents/Trafik Kazalarında tutulumunun Enerji Eşdeğer Hız Değerleri Tahmininde MFFNN ve GRNN Modelleri. *Int. J. Automot. Eng. Technol.* **2015**, *4*, 102. [CrossRef]
18. Park, S.H.; Kim, B.; Kang, C.M.; Chung, C.C.; Choi, J.W. Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture. *IEEE Intell. Veh. Symp. Proc.* **2018**, *2018*, 1672–1678. [CrossRef]
19. Savage, J.; Munoz, S.; Matamoros, M.; Osorio, R. Obstacle avoidance behaviors for mobile robots using genetic algorithms and recurrent neural networks. *IFAC Proc.* **2013**, *6*, 141–146. [CrossRef]
20. Li, F.; Gui, Z.; Zhang, Z.; Peng, D.; Tian, S.; Yuan, K.; Sun, Y.; Wu, H.; Gong, J.; Lei, Y. A hierarchical temporal attention-based LSTM encoder-decoder model for individual mobility prediction. *Neurocomputing* **2020**, *403*, 153–166. [CrossRef] [PubMed]
21. Wang, X.; Qin, Y.; Wang, Y.; Xiang, S.; Chen, H. ReLTanh: An activation function with vanishing gradient resistance for SAE-based DNNs and its application to rotating machinery fault diagnosis. *Neurocomputing* **2019**, *363*, 88–98. [CrossRef]
22. Krishnan, R.; Jagannathan, S.; Samaranayake, V.A. Direct Error Driven Learning for Deep Neural Networks with Applications to Bigdata. *Procedia Comput. Sci.* **2018**, *144*, 89–95. [CrossRef]
23. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1–32. [CrossRef] [PubMed]
24. Tanaka, M. Weighted sigmoid gate unit for an activation function of deep neural network. *Pattern Recognit. Lett.* **2020**, *135*, 354–359. [CrossRef]
25. Thakur, D. LSTM and Its Equations. 2018. Available online: https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af (accessed on 25 November 2020).
26. Wood, T. Softmax Function. n.d. Available online: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer (accessed on 25 November 2020).
27. Chang, Z.; Zhang, Y.; Chen, W. Electricity price prediction based on hybrid model of adam optimized LSTM neural network and wavelet transform. *Energy* **2019**, *187*, 115804. [CrossRef]
28. Fei, Z.; Wu, Z.; Xiao, Y.; Ma, J.; He, W. A new short-arc fitting method with high precision using Adam optimization algorithm. *Optik* **2020**, *212*, 164788. [CrossRef]
29. Kingma, D.P.; Ba, J.L. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings, San Diego, CA, USA, 7–9 May 2015.
30. Estefo, P.; Simmonds, J.; Robbes, R.; Fabry, J. The Robot Operating System: Package reuse and community dynamics. *J. Syst. Softw.* **2019**, *151*, 226–242. [CrossRef]
31. Sharifi, M.; Chen, X.Q.; Pretty, C.; Clucas, D.; Cabon-Lunel, E. Modelling and simulation of a non-holonomic omnidirectional mobile robot for offline programming and system performance analysis. *Simul. Model. Pract. Theory.* **2018**, *87*, 155–169. [CrossRef]
32. Koenig, N.; Howard, A. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, 28 September–2 October 2004; Available online: http://playerstage.sourceforge.net/gazebo/ (accessed on 18 May 2020).

33. Foundation, R. ROS. 2014. Available online: https://www.ros.org/ (accessed on 25 November 2020).
34. Open Source Robotics Foundation. Gazebo. 2014. Available online: http://gazebosim.org/ (accessed on 25 November 2020).
35. Python Software Foundation. Python. 2001. Available online: https://www.python.org/ (accessed on 25 November 2020).
36. GitHub Developers. Atom. n.d. Available online: https://atom.io/ (accessed on 25 November 2020).
37. E-Manual, R. TurtleBot3. n.d. Available online: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (accessed on 25 November 2020).
38. Lee, H.; Chae, H.; Yi, K. A Geometric Model based 2D LiDAR/Radar Sensor Fusion for Tracking Surrounding Vehicles. In *IFAC-PapersOnLine*; Elsevier: Amsterdam, The Netherlands, 2019; pp. 277–282. [CrossRef]
39. Oltean, S.E. Mobile Robot Platform with Arduino Uno and Raspberry Pi for Autonomous Navigation. *Procedia Manuf.* **2019**, *32*, 572–577. [CrossRef]
40. Shamsfakhr, F.; Sadeghibigham, B. A neural network approach to navigation of a mobile robot and obstacle avoidance in dynamic and unknown environments. *Turk. J. Electr. Eng. Comput. Sci.* **2017**, *25*, 1629–1642. [CrossRef]
41. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
42. Soltani, A.R.; Tawfik, H.; Goulermas, J.Y.; Fernando, T. Path planning in construction sites: Performance evaluation of the dijkstra, A*, and GA search algorithms. *Adv. Eng. Inform.* **2002**, *16*, 291–303. [CrossRef]
43. Fox, D.; Burgard, W.; Thrun, S. The dynamic window approach to collision avoidance. *IEEE Robot. Autom. Mag.* **1997**, *4*, 23–33. [CrossRef]
44. Molinos, E.J.; Llamazares, Á.; Ocaña, M. Dynamic window-based approaches for avoiding obstacles in moving. *Robot. Auton. Syst.* **2019**, *118*, 112–130. [CrossRef]