mathematics

MDPI

Article

Estimating Travel Time for Autonomous Mobile Robots through Long Short-Term Memory

Alexandru Matei, Stefan-Alexandru Precup, Dragos Circa, Arpad Gellert * and Constantin-Bala Zamfirescu

Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu, 550025 Sibiu, Romania; alex.matei@ulbsibiu.ro (A.M.); stefan.precup@ulbsibiu.ro (S.-A.P.); dragos.circa@ulbsibiu.ro (D.C.); constantin.zamfirescu@ulbsibiu.ro (C.-B.Z.)
* Correspondence: arpad.gellert@ulbsibiu.ro

Abstract: Autonomous mobile robots (AMRs) are gaining popularity in various applications such as logistics, manufacturing, and healthcare. One of the key challenges in deploying AMR is estimating their travel time accurately, which is crucial for efficient operation and planning. In this article, we propose a novel approach for estimating travel time for AMR using Long Short-Term Memory (LSTM) networks. Our approach involves training the network using synthetic data generated in a simulation environment using a digital twin of the AMR, which is a virtual representation of the physical robot. The results show that the proposed solution improves the travel time estimation when compared to a baseline, traditional mathematical model. While the baseline method has an error of 6.12%, the LSTM approach has only 2.13%.

Keywords: travel time estimation; digital twin; simulation; LSTM; AMR

MSC: 68T40

Citation: Matei, A.; Precup, S.-A.; Circa, D.; Gellert, A.; Zamfirescu, C.-B. Estimating Travel Time for Autonomous Mobile Robots through Long Short-Term Memory. *Mathematics* **2023**, *11*, 1723. https://doi.org/10.3390/math11071723

Academic Editors: Xiang Li and Ivan Lorencin

Received: 15 March 2023
Revised: 28 March 2023
Accepted: 31 March 2023
Published: 4 April 2023

1. Introduction

Autonomous mobile robots (AMRs) have become increasingly prevalent in various industries due to their ability to automate tasks and improve efficiency. AMRs are used in applications such as manufacturing, logistics, healthcare, and even in households. AMRs are more intelligent when compared to Automated Guided Vehicles (AGVs), as they are using simultaneous localization and mapping (SLAM) to navigate freely around the environment. Usually equipped with Lidar sensors and stereo cameras, they perceive the environment, planning paths and trajectories in near real-time. Their powerful onboard processing power allows them to be able to avoid stationary or mobile obstacles, have precise movement, and take decisions independent of a human operator. The ability to estimate travel time accurately is a critical factor in the performance of AMRs. Accurate travel time estimation can help AMRs to plan their paths and schedules, avoid collisions, and improve their overall performance. Estimation of travel time is also closely related to estimating the energy consumption, as based on the estimated mission time and travel speed, the robot must decide if it can complete the mission and still have enough energy and time respectively to be able to return to a charging station when necessary. Strategies employed in dynamic power management and energy consumption models include having different control schemes by disabling optional sensors [1], reducing the frequency of sensor reading, or offloading heavy processing tasks on a remote workstation. Such actions can determine suboptimal path planning, localization problems, reduced moving speed, and increased reaction time of the robot as it must wait for frequent data transfers.

The traditional approach to travel time estimation for AMRs involves using mathematical models, which are based on the physical characteristics of the robot and the environment. However, these models are often limited by the assumptions made during their development, and they may not be able to capture the complexities of the real-world

scenarios. Additionally, the performance of these models may degrade over time due to changes in the environment or the robot's condition.

The original contribution of this work is to analyze the performance of Long Short-Term Memory (LSTM), a type of recurrent neural network (RNN) that is specifically designed to process sequential data, for travel time estimation of autonomous mobile robots in a known environment, such as a factory work floor or other industrial environments. However, one of the challenges of using machine learning approaches such as LSTM for travel time estimation is the availability of data. Collecting large amounts of real-world data can be time-consuming and expensive. Another contribution is the creation of a synthetic dataset using simulation for travel time estimation. To increase the fidelity of the problem when compared to a real-world use case, we create stationary, randomly placed obstacles on the robot's path that can represent human workers, other mobile robots, or temporary placed objects.

The rest of the paper is organized as follows. Section 2 reviews the latest developments in travel time estimation and synthesizes some techniques that can be employed in this context, together with applications of LSTM networks. Section 3 describes the problem and presents the proposed LSTM-based travel time estimation solution while Section 4 evaluates the model and discusses the results. Finally, Section 5 concludes the paper and provides further work directions.

2. Related Work

Travel time estimation is mostly used in traffic management systems and traveler information systems with smart city and smart mobility applications. In this case, the data is acquired from several types of sensors such as loop detectors, magnetic sensors, cameras, radar, Global Positioning System (GPS), or passenger Global System for Mobile communication (GSM) signals. The developed solutions are applied to a specific area such as a city, certain highways, or specific routes and vehicles. Another important aspect of traffic data in real-world settings is that it is highly influenced by people's needs and behaviors, making it highly seasonal in respect to the day of the week or hour of the day, and other environmental parameters such as weather and temperature [2]. Authors of [3] conducted a review of data fusion methods for real-time traffic flow analysis which also include the travel time of the vehicles among other variables such as vehicle speed, distance, velocity, pedestrians, occupancy, weather, vehicle count, cellphone data, acceleration, lane behavior, temperature, density, wind speed, outgoing flow, trajectory, congestion, and incidents. Most of the methods from the articles analyzed in [3] were evaluated on datasets gathered using probe vehicles, GPS, or loop detectors from taxis, buses, freeways, and highways. As frequently used algorithms, the authors of [3] identified k-Nearest Neighbors (KNN), Bayesian, Gaussian Mixture Model (GMM), and other data fusion techniques. Next, we will detail some travel time estimation methods used in real-world applications.

A travel time estimation method based on vehicle trajectories is proposed in [4]. Their method consists of two steps: the first one estimates the travel time for each road segment in different time slots that were not traversed using context-aware tensor decomposition, and the second one optimally concatenates trajectories to estimate the travel time.

Another travel time estimation method based on generative adversarial networks (GAN) trained on trajectory data is presented in [5]. After the trajectory map matching step, pairs of road links and mini-trips are dynamically clustered by different traffic states using the Wasserstein distance algorithm. Finally, for each cluster, the resulting sub-dataset is used to train the proposed network, a Trip Information Maximizing GAN (T-InfoGAN).

An approach that uses just endpoint information data (timestamps and coordinates of origins and destination points) and distance and duration of the trip is presented in [6]. Their preprocessing step includes generating k-shortest paths for each origin–destination pair using map data, searching for the one that is closest to that specific pair trip distance. Using the processed dataset, a method called least-squares estimation with constraints is introduced that uses a probabilistic model for trip travel times.

Probability-based methods, artificial intelligence, and evidence theory-based methods are proposed in [7] to be used in wireless sensor networks when estimating travel time for traffic control in intelligent transportation systems.

In a traditional industrial environment, the manufacturing lines are fixed and inflexible. AGVs or AMRs are not used, and the transportation devices are operated by human drivers. With recent technological advances, shifting towards flexible manufacturing systems, the usage of AGVs have become widespread on factory shop floors. Still, the AGVs have limited path choices, being routed on predefined paths. Compared to traffic management systems that have real-world applications with characteristics such as seasonality, weather, or pedestrians present, manufacturing environments are controlled environments, optimizing to run at full capacity for as long as possible.

Authors of [8] introduced a novel multi-state scheduling algorithm that is applied in a flexible manufacturing environment. To improve the accuracy of their method, they used neural network-based travel time prediction. The network used is fully connected with one hidden layer, while the inputs considered for predictions are travel distance, number of turns, load status, and length of path that is in conflict both in the same and in the opposite direction.

In [9], a time estimation model is proposed for a rack-moving mobile robot system. The proposed quadratic mathematical model was used to find the optimum arrangement that would minimize the travel time while considering parameters such as the number of aisles, the number of layers, and layer depth.

Using a virtual space as a training environment is a common approach used in many areas. Advantages include reduced cost, higher speed as simulations can run faster than real-time, and improved safety for the human operators or the robot itself. Moreover, the information about the virtual environment that is used as input data by the network can be accessed in a much easier way compared to the real-world data. In [10], the authors used virtual space to train a deep reinforcement learning network to navigate in an environment without a map using low-dimensional range findings. Synthetic simulated data were also used for improving the efficiency of robot arm grasping action [11], fusing real and simulated data for vision-based autonomous flight [12], and even to process fault diagnosis [13].

LSTM is a network architecture that was developed to solve the vanishing/exploding gradient problem of RNN. Even though the architecture was proposed over 25 years ago in 1997 by Hochreiter and Schmidhuber [14] and then improved by adding a forget gate [15], LSTM networks are still popular and used in many applications to this day. LSTM networks have been applied in many fields, including financial sector [16,17], smart home [18], and energy consumption estimation [19,20]. On text data, LSTM was used for natural language processing tasks such as sign language to text translation [21], sentiment analysis [22], or text summarization [23]. In computer vision tasks, LSTM is usually paired with convolutional neural networks, so both spatial and temporal features are used for tasks such as action recognition [24], mode frequency detection [25], or video captioning [26]. On audio signals, LSTM was used for low-bitrate audio restoration [27], speech emotion recognition [28], or even to identify if a honeybee hive has a queen present or not [29].

In traffic analysis, LSTM is used for traffic flow prediction [30], pedestrian trajectory prediction [31], traffic congestion prediction [32], and traffic speed prediction [33]. LSTM is also used for path planning, risk assessment and mitigation in local path planning [34], path planning in unknown environments [35], and path planning for swarms of aerial unmanned vehicles [36].

In [37], authors studied the origin–destination passenger flow prediction in rail transit. They used an augmented LSTM neural network, called a Spatio–Temporal LSTM Network, that has a modified hidden layer structure to fuse the space and time dependencies. The model evaluation was performed on data of the Beijing subway network, acquired from multiple sources: origin–destination pairs, smart card data records, and mobile phone data.

Authors of [38] used a deep convolutional LSTM to predict city-wide taxi origin–destination flow. The input data is modeled as two matrices for travel time and travel

frequency that are both fed to the network at the same time. The model is evaluated on Xi'an City taxi trajectory data.

In [39], authors investigated the use of LSTM networks to model the assembly process of a configurable modular product. As the results show, the LSTM network was able to model all possible assembly scenarios compared to the previous implementations, which were not able to predict in the context of unseen scenarios.

Looking at the state of the art, we see that most travel time estimation articles are applied on smart city use cases: estimating the travel time for taxies, buses or cars on the freeways and highways. The environment for these use cases is rich with predictable and periodical events that can be used as network features: distance, velocity, pedestrians, weather, cellphone data, acceleration, lane behavior, temperature, density, wind speed, trajectory, congestion, and incidents. We see a few articles that study the estimation of travel time in an industrial environment. Since the usual industrial environments are strict and supervised, the paths that mobile robots can take are also fixed. Furthermore, almost all the features used in smart city applications are not present in an industrial environment. Because of this, the models used in industrial environments are left with a few selected features to work with. As such, one of the challenges when estimating travel time in industrial applications is to find the right features that the models can use to make accurate estimations.

Compared to previous articles applied to industrial environments [8,9], where AGVs with predefined paths are used, our use case estimates travel time for AMRs that are allowed to move freely in the environment. If we consider the methods used, approaches such as mathematical formulas, or fully connected neural networks are used, while our approach is different and evaluates an LSTM network.

3. Materials and Methods

3.1. Problem Description and Proposed Solution

The mobile robot travel time estimation is required in our case for accurate task planning, improved task allocation and reduced wait times. The task of the robot is to move prefabricated parts from one workstation to another as part of a reconfigurable manufacturing system detailed in [40]. The mobile robot that we are working with—Figure 1—has a detachable customizable platform on top for carrying the payload, four-wheel independent steering for increased mobility, and a Lidar sensor placed underneath for mapping and localization. The robot control is achieved using the Robot Operating System (ROS) [41], an open-source framework made specifically for creating robot applications.

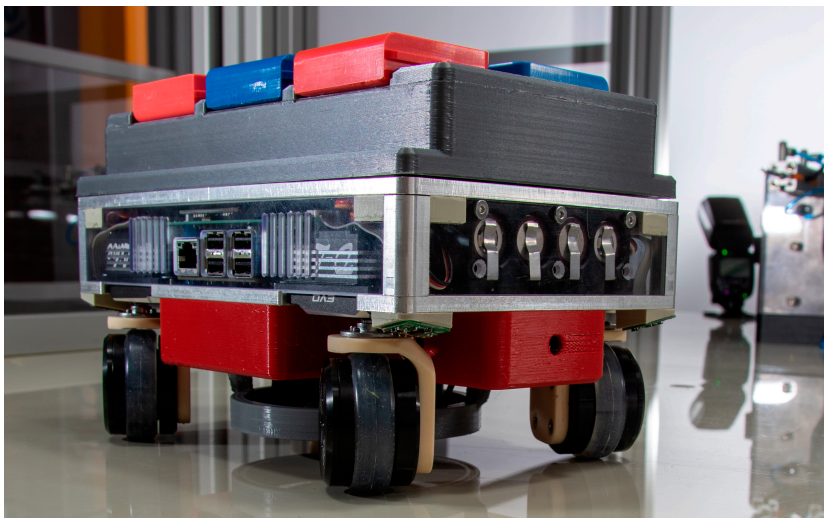


Figure 1. Mobile robot in the real world.

Considering a mathematical approach, we can naively estimate the travel time based on the expected travel distance multiplied by the speed of the robot, as in Equation (1), where Δx represents the distance and $\overline{v}$ is the speed. This is quite inaccurate since the speed is not constant: the robot has acceleration and deceleration time, in the real world the environment can contain zones with speed limits below the robot's cruise speed, and the robot's linear speed is different from the robot's angular speed. Additionally, when first computing the path plan, there is no information about possible dynamic obstacles on the way that can result in taking a different path at some point due to an obstacle, resulting in a different travel distance than the one that was used to calculate the travel time as follows:

$$t = \frac{\Delta x}{\overline{v}} \tag{1}$$

In this paper, we propose to use an LSTM network to estimate the travel time of the robot. This approach will be implemented as part of the robot fleet digital twin [42] that is currently under development. In this case, the digital twin is represented by the virtual replica of the robots deployed in the real world and is created by integrating various data sources, such as robot models and different sensors, that are used to monitor the performance, behavior, and the environment of the physical robots. Having that information, the digital twin can be used to analyze and optimize the performance of the robot fleet. Apart from the data integration and aggregation, the digital twin can also have features such as environment simulation to be able to test different scenarios and environment conditions, predictive analytics that use machine learning algorithms to analyze the current and historical real-world data to make predictions, remote control to test scenarios in the real world, and monitoring and analysis capabilities to view data in real time.

As such, we will make use of the environment simulation and predictive analytics features of the digital twin to simulate real-world scenarios for the robot and gather information about the travel time. The recorded simulation data will be used to train an LSTM network module that will predict the travel time. In the end, the same network can be used to predict using real-world data. One of the greatest advantages of using simulation data is that it allows for more control over the variables that affect the robots' travel time. This can lead to more accurate predictions and better performance when the algorithm is deployed in the real world.

When choosing the LSTM for our travel time estimation model, we also considered the computational complexity of the solution. For example, in a case where there are multiple types of AMR, a different model for each AMR type or even for each physical AMR should be trained. In this situation, the computational complexity increases linearly at most. A possible way to reduce the workload pressure on a single workstation is to distribute each model to its corresponding AMR if the hardware configuration allows it.

We used the previously mentioned mathematical approach as comparison for our proposed LSTM method, which is called naive in our graphs and tables. Since in our environment we do not have speed restrictions, in an ideal scenario, the robot will move at the max cruising speed of 0.22 m/s for the entire trip.

LSTM is a type of RNN that has become popular in recent years due to its ability to learn long-term dependencies and remember complex patterns. It can capture long-term dependencies in data, such as in a sequence of words, or even a time series of events. It can remember patterns in data over a long period of time, making them ideal for tasks that require long-term memory. For example, when predicting the next word in a sentence, an LSTM can remember words from earlier in the sentence and use this information to better predict the next word. Similarly, when predicting a time series of events, an LSTM can remember past events and use this information to better predict future events. LSTMs are used for many tasks such as natural language processing, language translation, speech recognition, and image classification.

LSTM cells are composed of several components, depicted in Figure 2. Each component is responsible for a specific function, such as memory, forget, input, and output.

The memory component (c_{t-1}) is responsible for remembering past information, the forget component is responsible for forgetting irrelevant information, the input component is responsible for taking in new information, the output component is responsible for providing an output based on the input and memory, and the input of the cell is composed of previously outputted data (h_{t-1}) and the current observation (x_t). All these components work together to learn long-term dependencies and remember complex patterns in data.

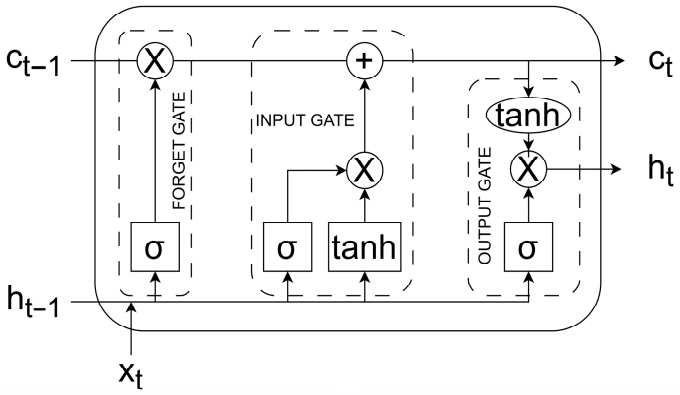


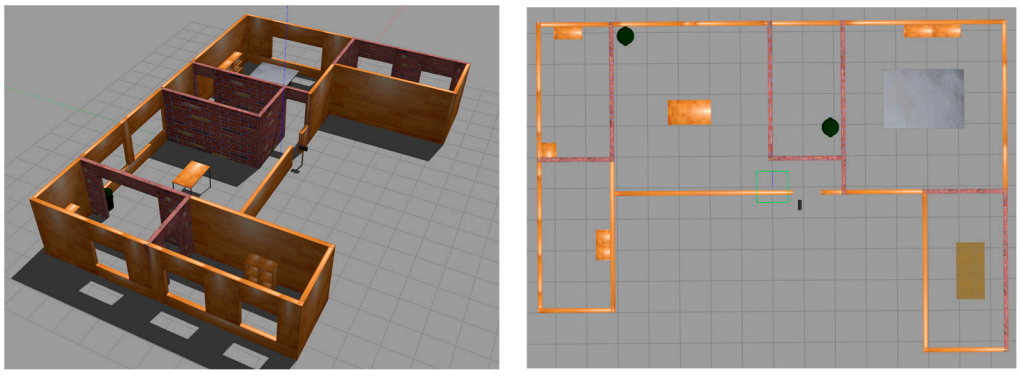
Figure 2. Internal structure of the LSTM cell.

3.2. Data Collection and Analysis

The current experiment aims to measure the time it takes for an autonomous mobile robot to get from point A to point B in a known environment, with dynamically spawned objects in the robot's way. We used a simulated environment to generate the training data, since running on a real robot inside a real-world environment would have taken a lot more time and human effort. As the real mobile robot is controlled using the ROS framework, we used Gazebo [43], an open-source simulator used for robot research that is well integrated with ROS.

3.2.1. Dataset Generation

The first step is to create a simulated environment. For replicability, we used the existing turtlebot3 house environment. Depicted in Figure 3, the environment is represented by a house of approximately 15 m $\times$ 10 m with several rooms and furniture. The environment is rather complex, as it has similar rooms, different door sizes, a hallway that can be a possible choke or deadlock zone, and tables that have a small footprint of four legs at the robot height level. The real robot is also modeled in Gazebo, and a few small changes are made to the existing robot control software packages to accommodate the new virtual sensors. The next step is to record the map of the environment using the default SLAM algorithm provided by ROS.



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

Figure 3. Simulated environment. (**a**) Perspective view. (**b**) Top-down view.

Having all the preparations ready, we developed an automated script that would start and record simulations of the robot traveling from one point to another. Each simulated sequence, as described in Figure 4, does the following actions: starts and initializes the required packages, sets the simulated environment to the default state, selects the current origin and destination points, moves the robot to the origin points, generates obstacles along the way, sets the robot's navigation goal to the destination point, and then waits for the robot to arrive at its destination. Next, we will detail these actions.

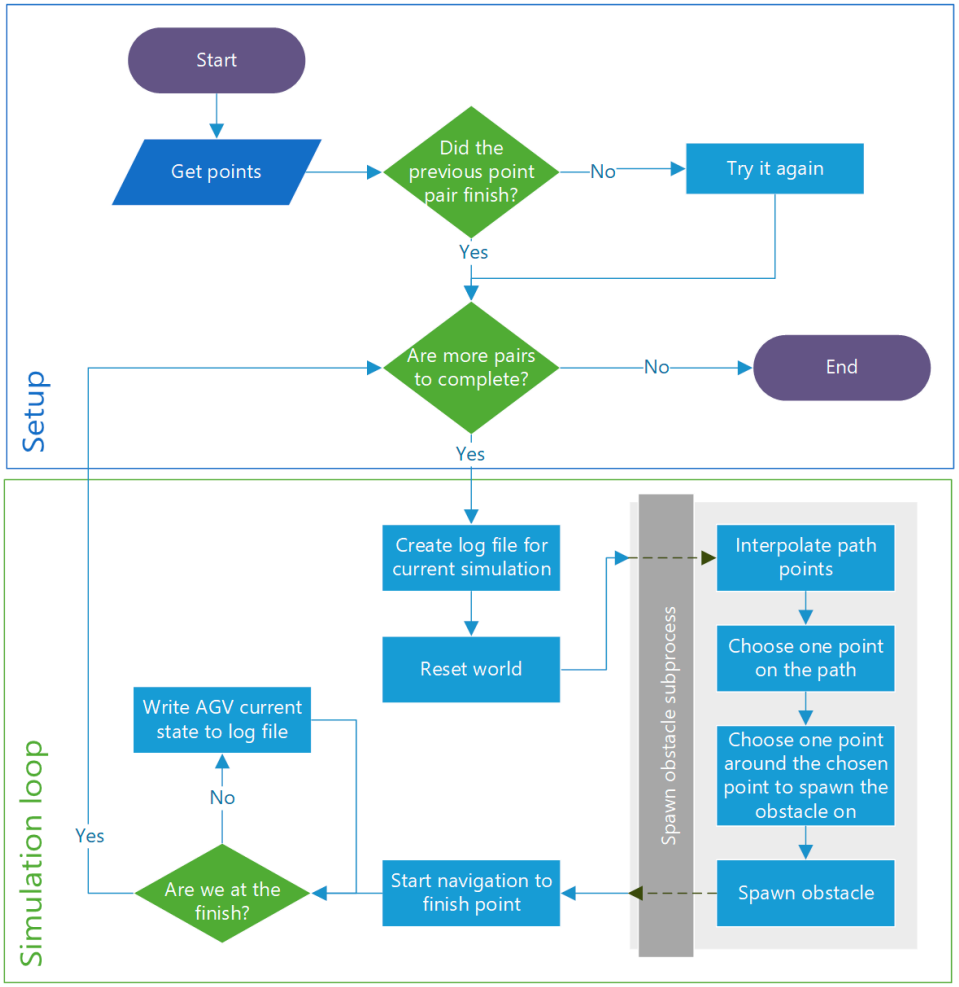


Figure 4. Automation program workflow.

The automated simulation script starts by launching the Gazebo simulator and loading the world environment. The robot model is loaded and the following required packages for robot navigation are started: sensor acquisition packages, *map_server* which provides the map of the environment that has been scanned beforehand, the *amcl* (Adaptive Monte Carlo Localization) package used for localization in a known map, the *move_base* package which provides an easy interface for the entire navigation stack packages (*global_planner*, *local_planner*, *global_costmap*, *local_costmap*, *recovery_behaviors*), and a *base_controller* package that translate generic movement commands to commands specific to the locomotion model of the robot. The next important step is to reset the simulation to the initial state if needed by removing any previously added obstacles and clearing the local and global cost maps. After this, the current origin and destination points are selected, and the robot is placed to the origin point coordinates. After this step, we can spawn an obstacle in the environment using the following steps: a temporary path plan is created between the current origin–destination points; a random point is selected from the list of points contained in the temporary path planned; using a normal distribution, displacements for both X and Y axis

are generated; the obstacle position is given by adding the coordinates of the randomly selected point in the path to the generated displacements. Of course, additional checks are made to be sure that the obstacle will not occupy the origin or destination point area, as objects can have various sizes and shapes. For simplicity, the obstacles are based on the 3D primitives available in Gazebo: box, sphere, and cylinder. We chose to spawn obstacles along the initial navigation planned path because spawning the objects randomly on the entire map would generate a lot of cases where the object would not influence the robot's travel time at all. Having spawned an obstacle object successfully, we can set the navigation goal of the robot to the destination point and wait for the simulation to finish. For every simulated sequence, we save the origin point and destination point pair, total travel time, and obstacle position. Furthermore, while the robot travels between the origin and destination points, the following information is saved periodically: current timestamp, robot position, global path, local path, and readings from the Lidar sensor, increasing the total number of examples in the dataset. Having 10 points chosen to cover the simulation environment, we had 90 possible scenarios that were simulated 100 times, each time with objects spawned in different places.

3.2.2. Dataset Analysis

Figure 5 shows the mapped environment with fixed obstacles (walls, furniture, table legs, and trash bins) detected in the mapping phase, drawn in black. The heat map in blue-red gradient represents the cost map associated with the environment map that is used by the path planning algorithm. A higher cost is assigned to areas close to obstacles, to ensure that the robot will most likely not collide with obstacles due to potential errors in navigation or localization. This will create paths that have minimum cost according to the cost map and not necessarily the shortest distance. The start and destination points that are used to generate the travel time estimation dataset are marked with red circles and have the following coordinates: {$[-6, -3]$, $[-6, 4]$, $[-3, 3.2]$, $[1, 2]$, $[5, 4]$, $[6, -5]$, $[-1, 4]$, $[-6, 0]$, $[6, 1]$, $[3, 4]$}.The origin of the coordinate system is marked with a yellow circle. The green lines represent the initial paths planned for the robot considering only the initial map, without any additional obstacles present.

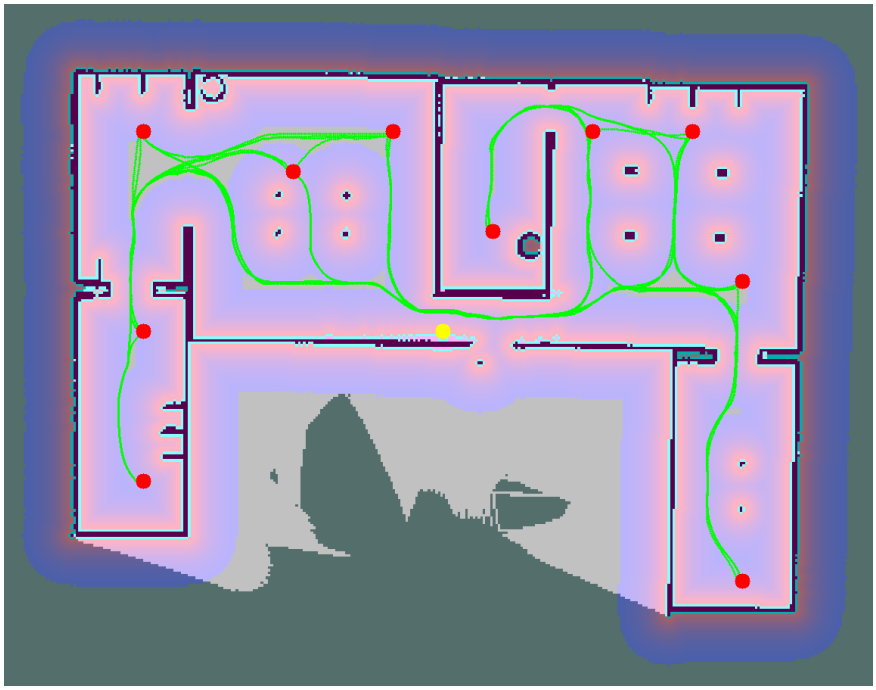


Figure 5. Map of the environment, cost map, start points and stop points marked with red, initial planned paths marked with green, and the origin of the coordinate system marked with yellow.

The position of generated obstacles—Figure 6—is based on the initial path planned for that specific scenario. As such, we see obstacle hotspots in places where initial paths overlap the most.

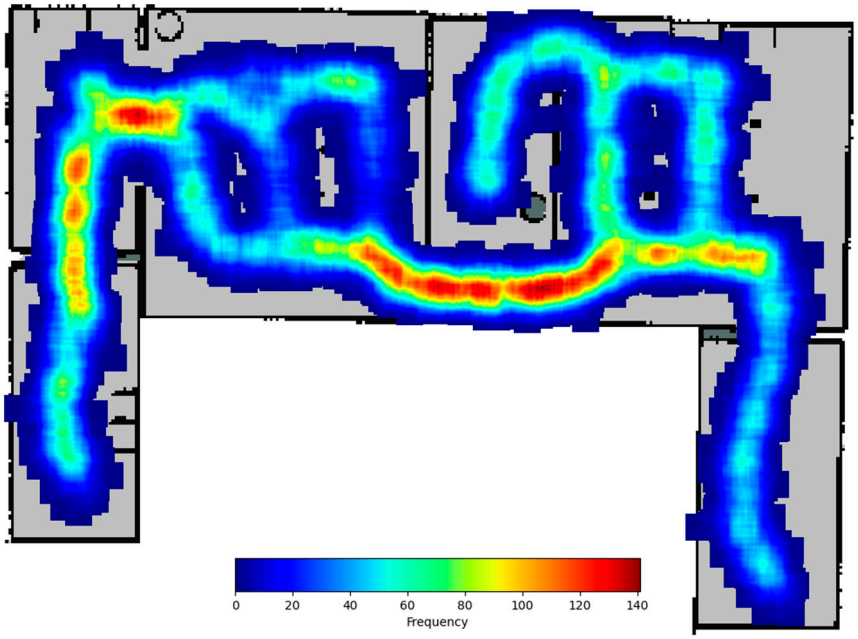


Figure 6. Map and heatmap of generated obstacles using a linear scale.

Figure 7 shows the map of the environment together with a heatmap of the mobile robot paths during the entire simulation using a logarithmic scale. It can be seen that the real trajectories that are updated taking into consideration the detected obstacles in the environment are broader and cover a larger area, when compared to the initial planned paths.

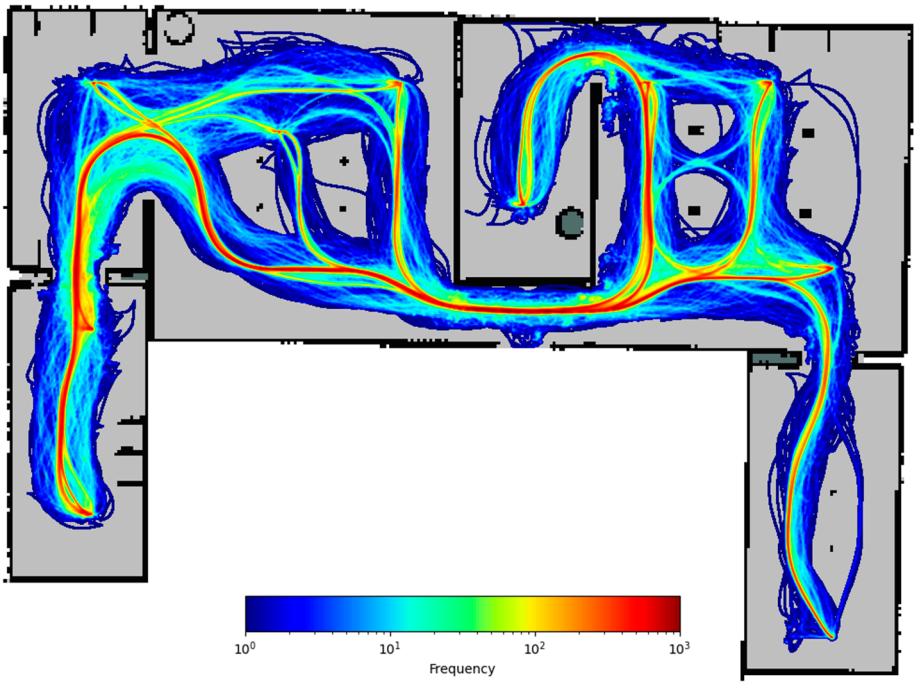


Figure 7. Map and heatmap of robot paths during simulation using logarithmic scale.

3.3. Hardware and Software Configurations

The computer used for simulation, network training, and evaluation is equipped with an Intel i7 9700KF CPU, 32 GB of DDR4 RAM, and an Nvidia RTX 2060 Ti GPU. The workstation runs Ubuntu 20.04. The simulation was performed using Gazebo 11.0 and ROS Noetic. All the software—simulation automation script, AMR software, dataset processing, and model training and evaluation scripts—were developed using Python together with the TensorFlow library.

4. Results

4.1. Experimental Methodology

Before effectively using the dataset for training, we had to preprocess it by applying multiple steps. The first step was to remove the simulations where the robot did not arrive at the destination regardless of the reason—usually these situations are caused by the robot getting stuck. At this step, we also removed the outliers on each subset of the origin–destination points by using the three-sigma rule. This operation reduced the number of simulations from 9000 to 8384, with histogram shown in Figure 8. This also ensured that the dataset did not contain abnormal or missing travel time data. We will note the dataset that contains only the initial travel time as the *initial_values* dataset. When taking into account the intermediate points of each origin–destination pair, there are approximately 568,000 entries, with histogram detailed in Figure 9, which will be noted as the *all_values* dataset. The final step is to enhance each dataset entry with the following computed variables: distance traveled from origin to current point, distance planned to travel from current point to destination based on the path planning, and the Euclidean distance to destination. The resulting dataset was split randomly into three subsets: 60% simulations for training, 20% simulations for validation, and 20% simulations for testing. The split percentages are chosen based on the result from Table 1.

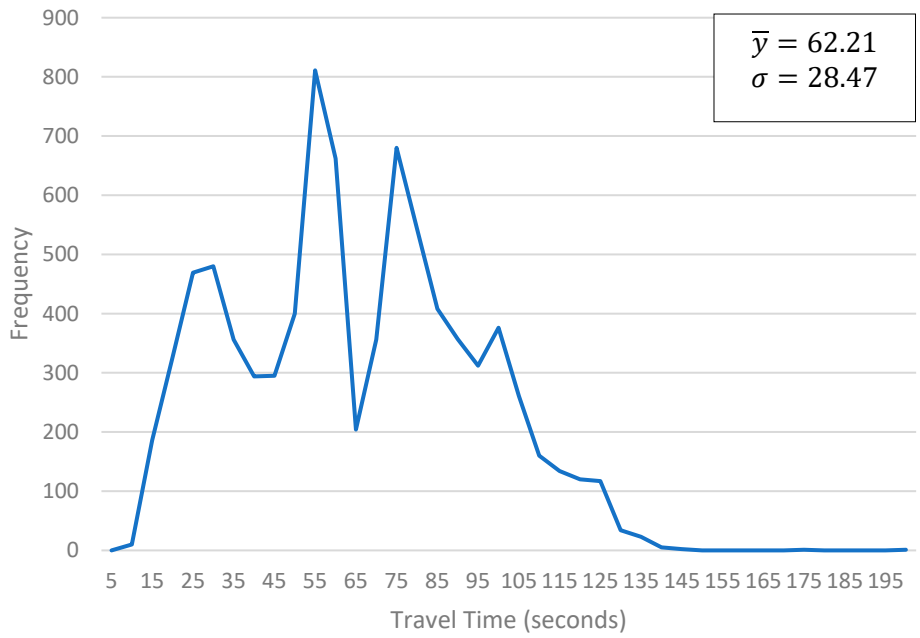


Figure 8. Histogram for *initial_values* dataset, mean and standard deviation.

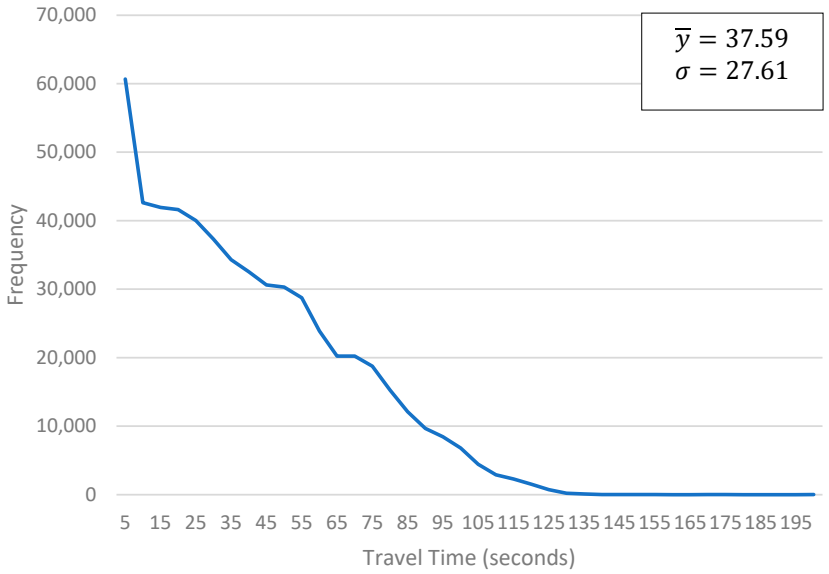


Figure 9. Histogram for *all_values* dataset, mean and standard deviation.

Table 1. Dataset split percentage.

	60-20-20	**70-15-15**	**80-10-10**	**90-5-5**
MAE	1.28	1.40	1.44	1.51
RMSE	2.65	3.67	3.51	3.26

The architecture of our model is presented in Figure 10. For the input layer, we used the following information from the dataset:

- Position {X, Y} and Target position {X, Y}.
- Distance left to target, computed based on Euclidean distance between current position and target position.
- Distance relative to origin, computed from AGV's {X, Y} position and map's origin point {0, 0}.
- Distance already traveled, computed based on previous position sequence: can contain condensed information about previously cleared obstacles.
- Distance planned to travel, computed based on the coordinates sequence present in the path planning: can contain condensed information about obstacles (if they are visible).

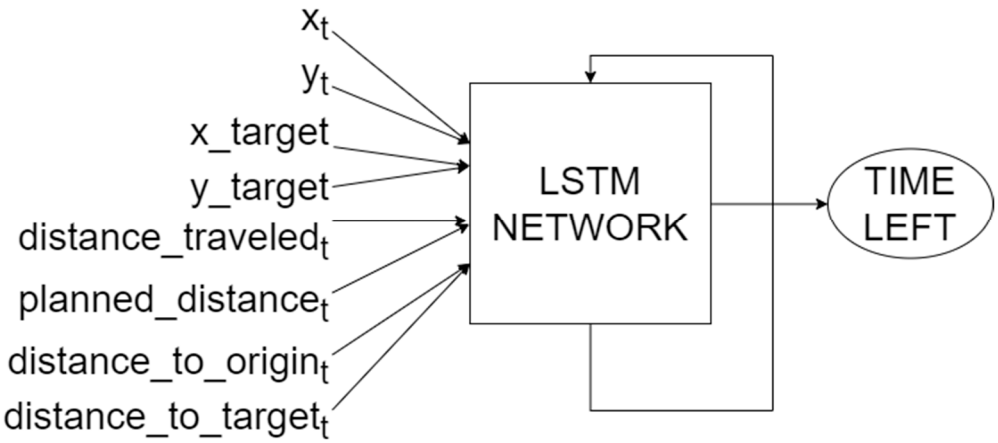


Figure 10. The proposed network architecture and input parameters.

All this information is fed to the LSTM network, which will make a decision based on the input data. The output data is a single cell, which will be the estimated time left for travel from the current position to the target position.

One feature that we took into consideration but did not include into the selected list of features was the Lidar scan data. As the Lidar sensor has 360° coverage, we have 360 readings, one for each degree. We tried to include no readings, and readings at every 1, 18, 24, 36, and 72 degrees. The results (Spreadsheet S5) show that the best choice is to not include any Lidar readings. One argument for this is that by adding another feature with up to 360 data elements, the network would require a wider and deeper configuration. Moreover, the Lidar information is already used when planning the current path from which we compute the planned distance input parameter.

The hyperparameters of the network structure that can be configured are the learning rate, dropout rate, the optimizer algorithm, number of LSTM layers, and number of LSTM units per layer. To find the optimal values, we performed hyperparameter tuning using the validation dataset. Based on the results from the Spreadsheets S1–S4 added as Supplementary Materials, the network training was done on the *all_values* training dataset using the Adamax optimizer on the Mean Absolute Error (MAE) loss function. The rest of the tuned hyperparameters have the following values: 0.01 learning rate, 0 dropout rate, and 1 LSTM layer with 8 units.

Model evaluation is done using the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Median Symmetric Accuracy (MSA) metrics. While the MAE value is directly proportional to the absolute error values, RMSE is more influenced by large error values. Compared to the popular Mean Absolute Percentage Error (MAPE) metric, MSA is symmetric to underprediction or overprediction, resistant to outliers, and can be interpreted as a percentage error [44]. In the formulas below, $\hat{Y}_i$ represents the predicted value, while Y_i is the actual true value.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|Y_i - \hat{Y}_i\right| \tag{2}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2} \tag{3}$$

$$\text{MSA} = 100\left(e^{(M(|\log_e(\frac{\hat{Y}_i}{Y_i})|))} - 1\right) \tag{4}$$

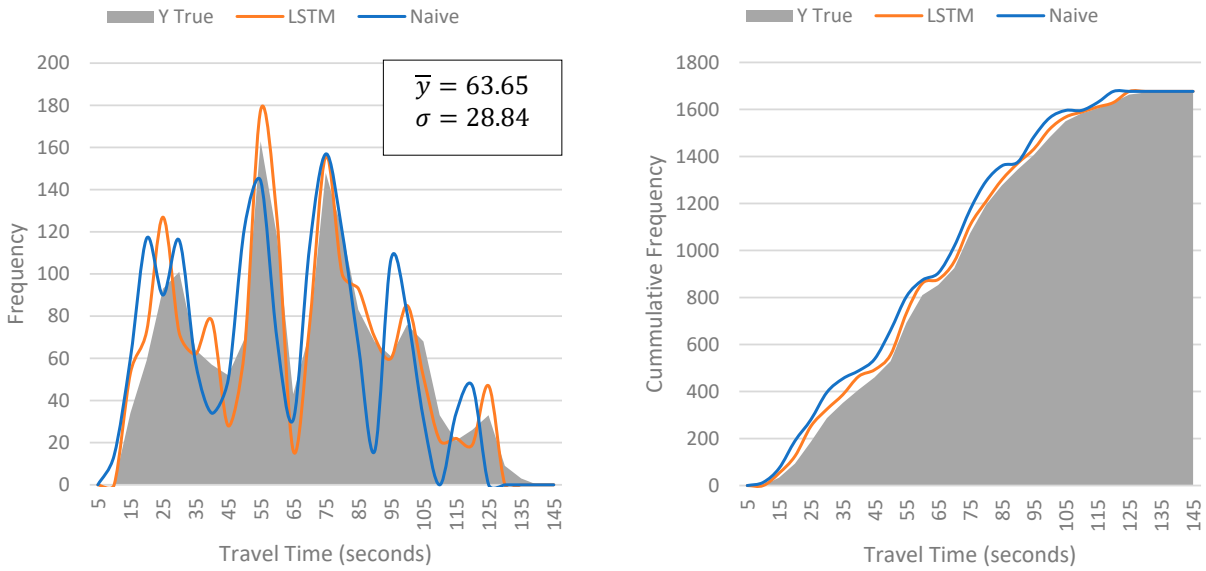
4.2. Evaluation

This section presents the model evaluation metrics MAE, RMSE, and MSA on the test datasets, compared to the Naive mathematical approach. The values of these metrics are in Table 2. At a glance, it can be easily seen that the LSTM approach is better than the Naive approach on all considered metrics for both *initial_values* and *all_values* datasets. MAE and RMSE, being scale-dependent metrics, are influenced by the multitude of values close to 0 that are present in the *all_values* dataset. This is not the case with the MSA metric, which is order-dependent and shows that the error magnitude is similar for both small and large values.

Table 2. Metrics of LSTM and Naive Time Formula.

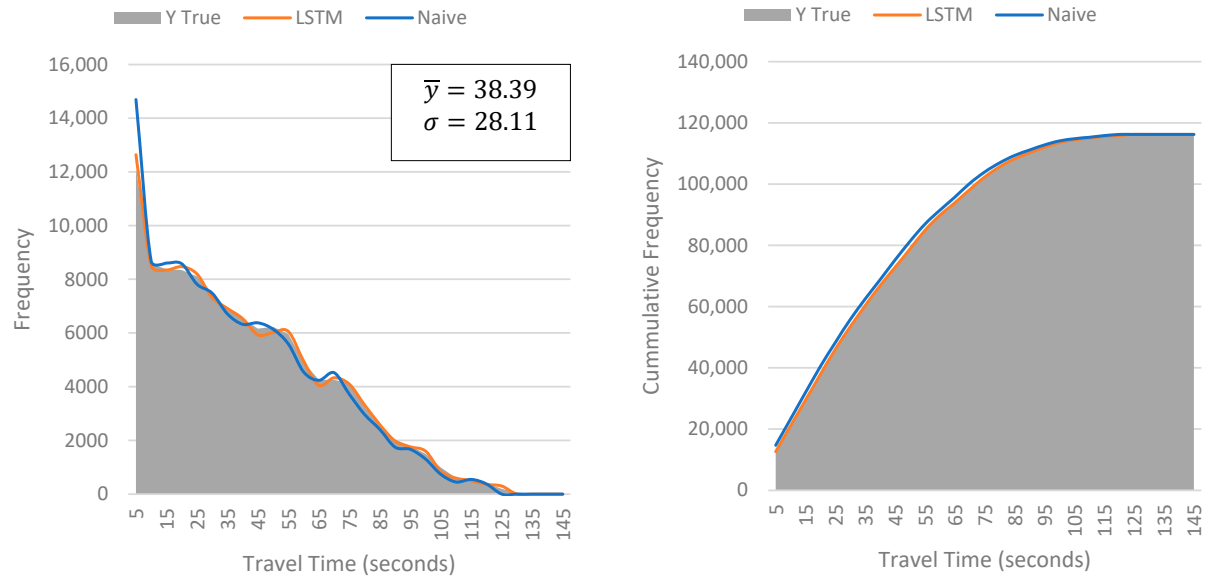
Data Set	LSTM			Naive		
	MAE	**RMSE**	**MSA**	**MAE**	**RMSE**	**MSA**
initial_values	3.38	8.02	2.56%	5.37	9.46	6.59%
all_values	1.29	2.64	2.13%	2.28	4.21	6.12%

Figures 11 and 12 show the frequency and cumulative frequency of the *initial_values* and *all_values* datesets, respectively. It is visible that the predicted values follow the real values distribution much better than the Naive approach, especially on the *initial_values* dataset. While the Naive approach shows deep valleys and high peaks compared to the true values, the LSTM approach follows the true values much closer. Both methods struggle in the lower interval (travel times between 0 and 50 s) on the *initial_values* dataset. We consider this to be caused by the multitude of training examples in that time interval present in the *all_values* dataset. The model is trained on sequences from the *all_values* dataset, while the *initial_values* dataset contains just the first element of each sequence.



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

Figure 11. *initial_values* dataset. (**a**) Frequency, mean and standard deviation. (**b**) Cumulative frequency.



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

Figure 12. *all_values* dataset. (**a**) Frequency, mean and standard deviation; (**b**) Cumulative frequency.

An important characteristic of the network is whether it is biased towards underpredicting or overpredicting, especially on the *initial_values* dataset as these values will influence the system planning the most. This is important because in an industrial setting with multiple robots and actions planned based on travel time prediction, an underprediction could cause a jam near a working station while an overprediction could allow the robot to anticipate it by waiting in designated areas along the way. In Figure 13, we can see that the Naive method underpredicts most of the time, while the LSTM approach is more centered and closer to 0. On the right side of Figure 13, the log accuracy ratio (LAR)—Equation (5)—of each prediction in the *initial_values* dataset is shown. Negative LAR values show an underprediction while a positive one shows an overprediction. Again, the LSTM method is more centered compared to the Naive method. The advantage of LAR is that it has a dimensionless value and can be compared across different datasets.

$$\mathrm{LAR} = \log_e\left(\frac{\hat{Y}_i}{Y_i}\right) \tag{5}$$

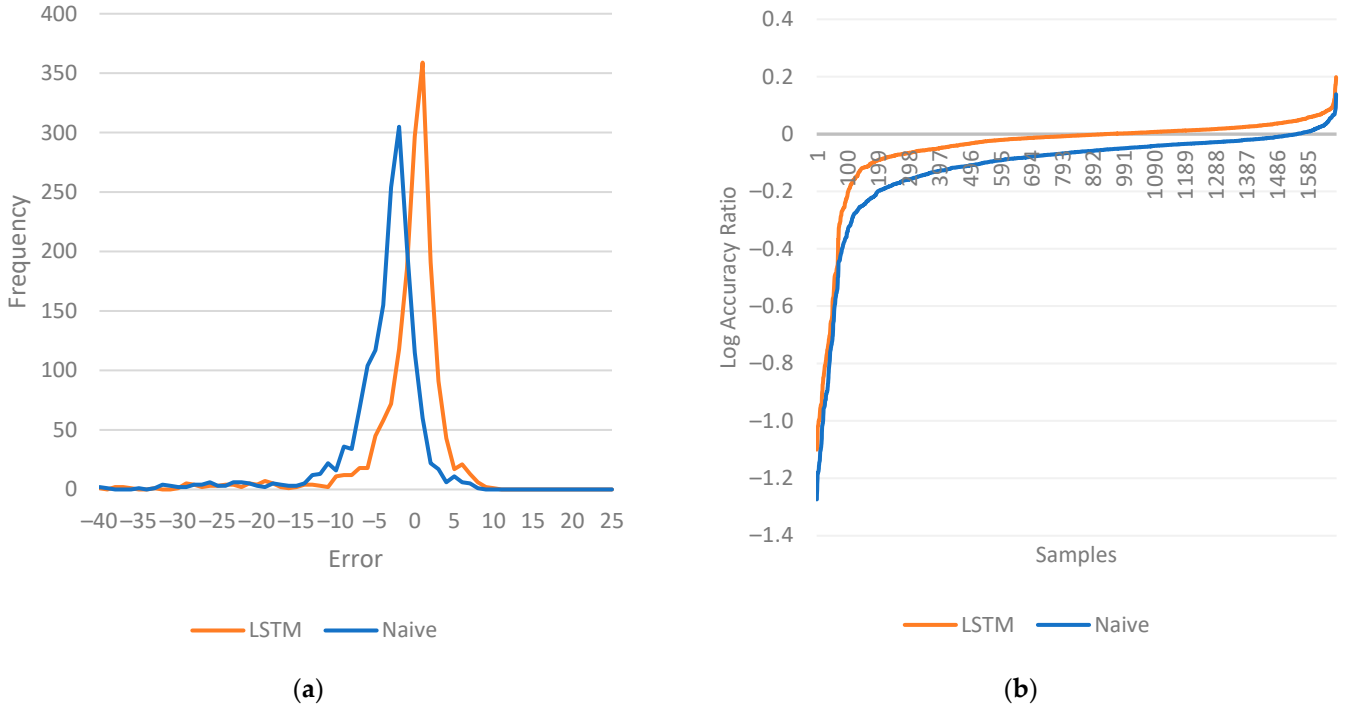
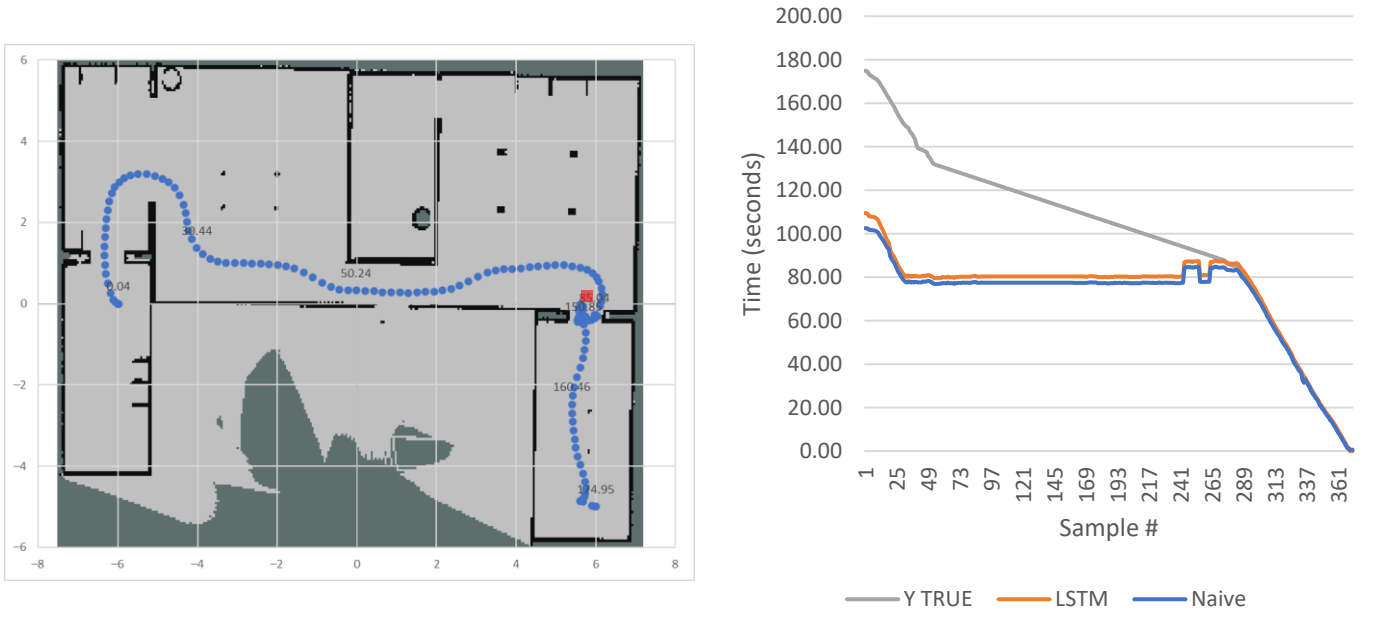


Figure 13. *initial_values* dataset. (**a**) Estimation error. (**b**) Log accuracy ratio.

4.3. Discussion

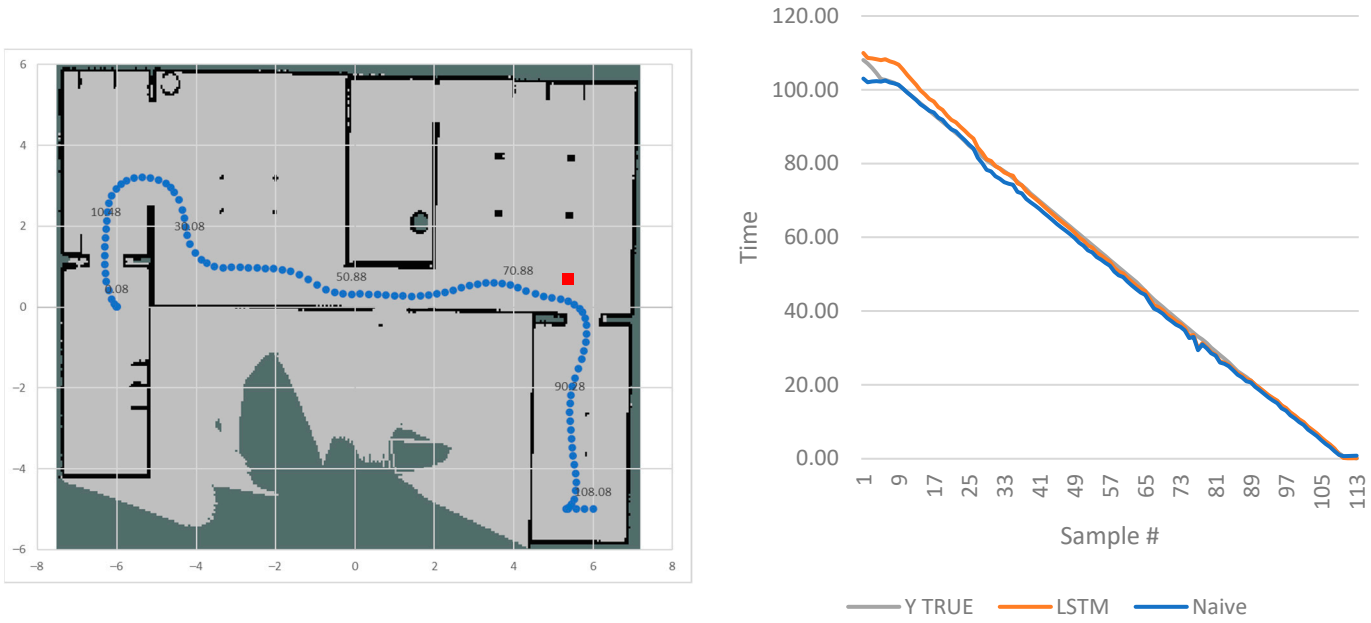
Figures 14 and 15 compare a case where the obstacle blocks a doorway and a usual case where an obstacle is placed in a spot that allows the robot to pass it easily. When looking at the inconvenient case, the big difference at the beginning between the actual time travel and the estimated one is due to the robot navigation behavior. Even though the robot finds a feasible path plan to overcome the obstacle, localization errors can pose difficulties in implementing that plan. Another explanation is that if the robot does not have a great positioning precision in the environment, there will be multiple trials of squeezing through the small gap. In addition, if the robot navigation stack detects a deadlock, there are several recovery behaviors that the robot can try to get out of the deadlock, such as a recovery behavior that rotates the robot in place attempting to move the obstacles around the robot, a recovery behavior that clears the navigation stack cost maps, or a recovery behavior that reduces the robot translation and rotational speed to increase the positioning precision. Of

course, these recovery strategies add unexpected delays. As soon as the obstacle is cleared, the estimations are very close to the real remaining travel time.



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

Figure 14. The path is blocked. (**a**) Trajectory on map. Red square marks the position of the obstacle. (**b**) Time samples and estimates of travel time.



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

Figure 15. The path is clear. (**a**) Trajectory on map. Red square marks the position of the obstacle. (**b**) Time samples and estimates of travel time.

Since in case of a possible deadlock, the default implemented ROS behavior is clearing it in a trial-and-error way and ultimately requesting human assistance after a timeout, we consider these extreme cases to be a limitation of the current used model and possibly of other approaches too.

Another aspect that should be discussed is the capability of the network to generalize in other situations. We consider that small changes to the environment will not affect the model or at least not in a considerable manner. In our specific environment, a new chair added to a room, or a piece of furniture moved to another place would be considered as any other situation with an obstacle on the path that was present in the training dataset. Of course, a totally new environment will be considered a major change and will require a new model to be trained.

An aspect that could be discussed refers to the steps and associated costs required to apply this research in a real-world scenario. In our opinion, the costliest and most time-consuming operation would be to create an accurate simulation of the factory that will allow the generation of the training datasets. Even if creating a simulation environment for the specific industrial use case is an expensive task, there are multiple ways in which it can be used, and not only for travel time estimation. If that organization will use the simulation for other use cases as well, then the cost associated only with travel time estimation will be lower. Another source for the dataset would be to use previous travel times if they were recorded. A risk with this second approach is that some of the recorded data is not accurate for the current configuration of the environment, whereas the AMR that generated the data is replaced or the layout is changed. We consider that the training of the network and inference cost to be much smaller compared to the cost of developing the simulation environment. Looking at the error difference of 3.99%, from 6.12% with the Naive approach to 2.13% when using the LSTM method, it might not be significant for a small number of AMRs, but for an organization with multiple locations and a much higher number of AMRs, it might be beneficial.

5. Conclusions

The original contribution of this work was applying LSTM for travel time estimation of AMRs. The article successfully shows that a neural network approach, in this case LSTM, gives better results in estimating the travel time of an autonomous robot when compared with the traditional way of computing time based on speed and distance when there is historical data available. The entire work is carried out by making use of digital twin concepts: the training data is generated in simulations, reducing the required effort, and the machine learning algorithm is implemented as part of the prediction and analytics module of the digital twin. After the algorithm is trained and tested, data gathered from the real world can be used to estimate the real travel time of the robot. Another thing to consider is that the features used as network input are limited compared to the use cases present in real-life situations such as public transport travel time, where parameters including weather, time of day, or temperature also influence the result. Still, the LSTM network manages to learn and estimate correctly with limited features.

There are several points that require our attention for further work. Currently the planned path information is reduced to the planned distance to travel parameter. We expect that by using the planned path, the network can determine if the chosen route has a higher chance of delays and adjust accordingly. Another research direction is to see if a more complex LSTM network, such as Bidirectional LSTM, or a combination of LSTM layer and other types of layers will provide a performance improvement and at what computational cost. In the long term, our goal is to incorporate multiple robots in the simulation, handle more sensor information, and simulate the demand for parts, which ultimately will improve the overall time estimation and efficiency of our task planning model.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/math11071723/s1, Spreadsheet S1: Learning Rate; Spreadsheet S2: Dropout; Spreadsheet S3: Layer 2—No. of LSTM units, Spreadsheet S4: Optimizer, Spreadsheet S5: Lidar scan interval, Spreadsheet S6: Final model results.

Author Contributions: Conceptualization, A.M., A.G. and C.-B.Z.; methodology, A.M., A.G., S.-A.P. and C.-B.Z.; software, A.M., S.-A.P. and D.C.; validation, A.M. and S.-A.P.; formal analysis, A.M., S.-A.P., A.G., D.C. and C.-B.Z.; investigation, A.M. and D.C.; resources, C.-B.Z.; data curation, A.M. and S.-A.P.; writing—original draft preparation, A.M., S.-A.P., A.G., D.C. and C.-B.Z.; writing—review and editing, A.M., S.-A.P., A.G., D.C. and C.-B.Z.; visualization, A.M., S.-A.P. and D.C.; supervision, A.G. and C.-B.Z.; project administration, C.-B.Z.; funding acquisition, C.-B.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Hasso Plattner Excellence Research Grant LBUS-HPI-ERG-2020-03, financed by the Knowledge Transfer Center of the Lucian Blaga University of Sibiu.

Data Availability Statement: The data presented in this study are available upon request.

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

References

1. Jaiem, L.; Druon, S.; Lapierre, L.; Crestani, D. A step toward mobile robots autonomy: Energy estimation models. In *Towards Autonomous Robotic Systems: 17th Annual Conference, TAROS 2016, Sheffield, UK, 26 June–1 July 2016*; Proceedings 17; Springer: Cham, Switzerland, 2016; pp. 177–188. [CrossRef]
2. Zhang, D.; Kabuka, M.R. Combining weather condition data to predict traffic flow: A GRU-based deep learning approach. *IET Intell. Transp. Syst.* **2018**, *12*, 578–585. [CrossRef]
3. Kashinath, S.A.; Mostafa, S.A.; Mustapha, A.; Mahdin, H.; Lim, D.; Mahmoud, M.A.; Mohammed, M.A.; Al-Rimy, B.A.; Fudzee, M.F.; Yang, T.J. Review of data fusion methods for real-time and multi-sensor traffic flow analysis. *IEEE Access* **2021**, *9*, 51258–51276. [CrossRef]
4. Wang, Y.; Zheng, Y.; Xue, Y. Travel time estimation of a path using sparse trajectories. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 24–27 August 2014; pp. 25–34. [CrossRef]
5. Zhang, K.; Jia, N.; Zheng, L.; Liu, Z. A novel generative adversarial network for estimation of trip travel time distribution with trajectory data. *Transp. Res. Part C Emerg. Technol.* **2019**, *108*, 223–244. [CrossRef]
6. Mridha, S.; Ganguly, N.; Bhattacharya, S. Link travel time prediction from large scale endpoint data. In Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Redondo Beach, CA, USA, 7–10 November 2017; pp. 1–4. [CrossRef]
7. Guido, G.; Haghshenas, S.S.; Vitale, A.; Astarita, V. Challenges and opportunities of using data fusion methods for travel time estimation. In Proceedings of the 2022 8th International Conference on Control, Decision and Information Technologies (CoDIT), Istanbul, Turkey, 17–20 May 2022; Volume 1, pp. 587–592. [CrossRef]
8. Wang, X.; Wu, W.; Xing, Z.; Chen, X.; Zhang, T.; Niu, H. A neural network based multi-state scheduling algorithm for multi-AGV system in FMS. *J. Manuf. Syst.* **2022**, *64*, 344–355. [CrossRef]
9. Wang, K.; Yang, Y.; Li, R. Travel time models for the rack-moving mobile robot system. *Int. J. Prod. Res.* **2020**, *58*, 4367–4385. [CrossRef]
10. Tai, L.; Paolo, G.; Liu, M. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 31–36. [CrossRef]
11. Bousmalis, K.; Irpan, A.; Wohlhart, P.; Bai, Y.; Kelcey, M.; Kalakrishnan, M.; Downs, L.; Ibarz, J.; Pastor, P.; Konolige, K.; et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In Proceedings of the 2018 IEEE international conference on robotics and automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 4243–4250. [CrossRef]
12. Kang, K.; Belkhale, S.; Kahn, G.; Abbeel, P.; Levine, S. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 6008–6014. [CrossRef]
13. Li, W.; Gu, S.; Zhang, X.; Chen, T. Transfer learning for process fault diagnosis: Knowledge transfer from simulation to physical processes. *Comput. Chem. Eng.* **2020**, *139*, 106904. [CrossRef]
14. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef]
15. Gers, F.A.; Schmidhuber, J.; Cummins, F. Learning to forget: Continual prediction with LSTM. *IET Conf. Proc.* **1999**, *2*, 850–855. [CrossRef]
16. Bhandari, H.N.; Rimal, B.; Pokhrel, N.R.; Rimal, R.; Dahal, K.R.; Khatri, R.K. Predicting stock market index using LSTM. *Mach. Learn. Appl.* **2022**, *9*, 100320. [CrossRef]
17. Liang, Y.; Lin, Y.; Lu, Q. Forecasting gold price using a novel hybrid model with ICEEMDAN and LSTM-CNN-CBAM. *Expert Syst. Appl.* **2022**, *206*, 117847. [CrossRef]
18. Mekruksavanich, S.; Jitpattanakul, A. LSTM Networks Using Smartphone Data for Sensor-Based Human Activity Recognition in Smart Homes. *Sensors* **2021**, *21*, 1636. [CrossRef]

19. Huang, X.; Li, Q.; Tai, Y.; Chen, Z.; Liu, J.; Shi, J.; Liu, W. Time series forecasting for hourly photovoltaic power using conditional generative adversarial network and Bi-LSTM. *Energy* **2022**, *246*, 123403. [CrossRef]
20. Bachici, M.A.; Gellert, A. Modeling Electricity Consumption and Production in Smart Homes using LSTM Networks. *Int. J. Adv. Stat. IT&C Econ. Life Sci.* **2020**, *10*, 80–89.
21. Xiao, Q.; Chang, X.; Zhang, X.; Liu, X. Multi-information spatial–temporal LSTM fusion continuous sign language neural machine translation. *IEEE Access* **2020**, *8*, 216718–216728. [CrossRef]
22. Jelodar, H.; Wang, Y.; Orji, R.; Huang, S. Deep sentiment classification and topic discovery on novel coronavirus or COVID-19 online discussions: NLP using LSTM recurrent neural network approach. *IEEE J. Biomed. Health Inform.* **2020**, *10*, 2733–2742. [CrossRef]
23. Jiang, J.; Zhang, H.; Dai, C.; Zhao, Q.; Feng, H.; Ji, Z.; Ganchev, I. Enhancements of Attention-Based Bidirectional LSTM for Hybrid Automatic Text Summarization. *IEEE Access* **2021**, *9*, 123660–123671. [CrossRef]
24. Ullah, A.; Ahmad, J.; Muhammad, K.; Sajjad, M.; Baik, S.W. Action recognition in video sequences using deep bi-directional LSTM with CNN features. *IEEE Access* **2017**, *6*, 1155–1166. [CrossRef]
25. Yang, R.; Singh, S.K.; Tavakkoli, M.; Amiri, N.; Yang, Y.; Karami, M.A.; Rai, R. CNN-LSTM deep learning architecture for computer vision-based modal frequency detection. *Mech. Syst. Signal Process.* **2020**, *144*, 106885. [CrossRef]
26. Gao, L.; Guo, Z.; Zhang, H.; Xu, X.; Shen, H.T. Video captioning with attention-based LSTM and semantic consistency. *IEEE Trans. Multimed.* **2017**, *19*, 2045–2055. [CrossRef]
27. Deng, J.; Schuller, B.; Eyben, F.; Schuller, D.; Zhang, Z.; Francois, H.; Oh, E. Exploiting time-frequency patterns with LSTM-RNNs for low-bitrate audio restoration. *Neural Comput. Appl.* **2020**, *32*, 1095–1107. [CrossRef]
28. Wang, J.; Xue, M.; Culhane, R.; Diao, E.; Ding, J.; Tarokh, V. Speech emotion recognition with dual-sequence LSTM architecture. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 6474–6478. [CrossRef]
29. Ruvinga, S.; Hunter, G.J.; Duran, O.; Nebel, J.C. Use of LSTM networks to identify "queenlessness" in honeybee hives from audio signals. In Proceedings of the 2021 17th International Conference on Intelligent Environments, Dubai, United Arab Emirates, 21–24 June 2021; pp. 1–4. [CrossRef]
30. Wang, Z.; Su, X.; Ding, Z. Long-term traffic prediction based on LSTM encoder-decoder architecture. *IEEE Trans. Intell. Transp. Syst.* **2020**, *22*, 6561–6571. [CrossRef]
31. Xue, H.; Huynh, D.Q.; Reynolds, M. PoPPL: Pedestrian trajectory prediction by LSTM with automatic route class clustering. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 77–90. [CrossRef]
32. Ranjan, N.; Bhandari, S.; Zhao, H.P.; Kim, H.; Khan, P. City-wide traffic congestion prediction based on CNN, LSTM and transpose CNN. *IEEE Access* **2020**, *8*, 81606–81620. [CrossRef]
33. Zafar, N.; Haq, I.U.; Chughtai, J.-U.-R.; Shafiq, O. Applying Hybrid Lstm-Gru Model Based on Heterogeneous Data Sources for Traffic Speed Prediction in Urban Areas. *Sensors* **2022**, *22*, 3348. [CrossRef]
34. Wang, H.; Lu, B.; Li, J.; Liu, T.; Xing, Y.; Lv, C.; Cao, D.; Li, J.; Zhang, J.; Hashemi, E. Risk assessment and mitigation in local path planning for autonomous vehicles with LSTM based predictive model. *IEEE Trans. Autom. Sci. Eng.* **2021**, *19*, 2738–2749. [CrossRef]
35. Yao, H.; Liu, Y.; Zhang, X. Developing deep LSTM model for real-time path planning in unknown environments. In Proceedings of the 2020 7th International Conference on Dependable Systems and Their Applications (DSA), Xi'an, China, 28–29 November 2020; pp. 219–225. [CrossRef]
36. Schlichting, M.R.; Notter, S.; Fichter, W. LSTM-based spatial encoding: Explainable path planning for time-variant multi-agent systems. In Proceedings of the AIAA Scitech 2021 Forum, Virtual, 11–15 & 19–21 January 2021; p. 1860. [CrossRef]
37. Li, D.; Cao, J.; Li, R.; Wu, L. A spatio-temporal structured LSTM model for short-term prediction of origin-destination matrix in rail transit with multisource data. *IEEE Access* **2020**, *8*, 84000–84019. [CrossRef]
38. Duan, Z.; Zhang, K.; Chen, Z.; Liu, Z.; Tang, L.; Yang, Y.; Ni, Y. Prediction of city-scale dynamic taxi origin-destination flows using a hybrid deep neural network combined with travel time. *IEEE Access* **2019**, *7*, 127816–127832. [CrossRef]
39. Precup, S.-A.; Gellert, A.; Dorobantiu, A.; Zamfirescu, C.-B. Assembly process modeling through long short-term memory. In *Recent Challenges in Intelligent Information and Database Systems: 13th Asian Conference, ACIIDS 2021, Phuket, Thailand, 7–10 April 2021*; Springer: Singapore, 2021; pp. 28–39. [CrossRef]
40. Matei, A.; Pirvu, B.-C.; Petruse, R.E.; Candea, C.; Zamfirescu, B.-C. Designing a Multi-agent Control System for a Reconfigurable Manufacturing System. In *Service Oriented, Holonic and Multi-Agent Manufacturing Systems for Industry of the Future: Proceedings of SOHOMA 2022, Bucharest, Romania, 22–23 September 2022*; Springer: Cham, Switzerland, 2023; pp. 434–445. [CrossRef]
41. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3, p. 5.
42. Matei, A.; Circa, D.; Zamfirescu, B.-C. Digital Twin for automated guided vehicles fleet management. *Procedia Comput. Sci.* **2022**, *199*, 1363–1369. [CrossRef]

43. 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 (IROS), Sendai, Japan, 28 September–2 October 2004; Volume 3, pp. 2149–2154. [CrossRef]
44. Morley, S.K.; Brito, T.V.; Welling, D.T. Measures of model performance based on the log accuracy ratio. *Space Weather* **2018**, *16*, 69–88. [CrossRef]