robotics

Article

A Vision Dynamics Learning Approach to Robotic Navigation in Unstructured Environments

Cosmin Ginerica [1,*], Mihai Zaha [1], Laura Floroian [1], Dorian Cojocaru [2] and Sorin Grigorescu [1]

[1] Robotics, Vision and Control Laboratory (ROVIS), Transilvania University of Brasov, 500036 Brasov, Romania; mihai.zaha@unitbv.ro (M.Z.); lauraf@unitbv.ro (L.F.); s.grigorescu@unitbv.ro (S.G.)
[2] Electronics and Mechatronics, Department of Automatic Control, University of Craiova, 200585 Craiova, Romania; cojocaru@robotics.ucv.ro
[*] Correspondence: cosmin.ginerica@unitbv.ro

Abstract: Autonomous legged navigation in unstructured environments is still an open problem which requires the ability of an intelligent agent to detect and react to potential obstacles found in its area. These obstacles may range from vehicles, pedestrians, or immovable objects in a structured environment, like in highway or city navigation, to unpredictable static and dynamic obstacles in the case of navigating in an unstructured environment, such as a forest road. The latter scenario is usually more difficult to handle, due to the higher unpredictability. In this paper, we propose a vision dynamics approach to the path planning and navigation problem for a quadruped robot, which navigates in an unstructured environment, more specifically on a forest road. Our vision dynamics approach is based on a recurrent neural network that uses an RGB-D sensor as its source of data, constructing sequences of previous depth sensor observations and predicting future observations over a finite time span. We compare our approach with other state-of-the-art methods in obstacle-driven path planning algorithms and perform ablation studies to analyze the impact of architectural changes to our model components, demonstrating that our approach achieves superior performance in terms of successfully generating collision-free trajectories for the intelligent agent.

Keywords: deep learning; planning; recurrent neural network; robotics

check for updates

Citation: Ginerica, C.; Zaha, M.; Floroian, L.; Cojocaru, D.; Grigorescu, S. A Vision Dynamics Learning Approach to Robotic Navigation in Unstructured Environments. *Robotics* **2024**, *13*, 15. https://doi.org/10.3390/robotics13010015

Academic Editors: Hong Zhang and Marco Ceccarelli

Received: 3 December 2023
Revised: 2 January 2024
Accepted: 15 January 2024
Published: 17 January 2024

1. Introduction

One of the most important sub-tasks of autonomous navigation, especially navigation bound to an unstructured environment, is an intelligent agent's ability to perform collision-free path planning. It is paramount that the robot's surroundings are understood to its best ability, such that it can safely navigate in difficult environments. Traditional approaches rely on heuristic rules of handcrafted features for solving the path planning task. However, these methods may not extend well enough in the context of dynamic environments.

With the accelerated development of computing hardware, the Artificial Intelligence domain has also seen an enormous rise in popularity. As a consequence, an alternative approach to self-driving tasks is based on AI algorithms, commonly implemented as deep neural networks. These algorithms, which are usually trained using sensory data acquired from the autonomous agent, show potential in generalizing to new and/or evolving environments.

Some AI-based approaches to the autonomous driving challenge divide the navigation problem into smaller sub-problems and approach them individually. Others model the entire navigation task as a black-box, expecting sensor data inputs and directly outputting vehicle commands. Some of these algorithms use RGB image data in order to detect obstacles in the vicinity of the autonomous agent and plan collision-free trajectories based on the detected obstacles.

In this paper, we propose a different approach for tackling the autonomous navigation problem, by using a recurrent neural network to model the dynamics of the environment in which the robot navigates. The environment dynamics, in the context of this work, refer to dynamic obstacles present in the scene. Recurrent neural networks are a type of model that can capture temporal dependencies and process sequence data, like objects moving within the scene. By incorporating historic data in the form of previous sensor observations from the mobile robot, a Recurrent Neural Network (RNN) can improve the accuracy and robustness of a collision prediction algorithm, especially in complex, unstructured environments.

We have chosen a recurrent neural network architecture due to its ability to capture temporal dependencies within the input data. This enables the encoding of obstacle dynamics within its layers and allows for using this additional information in the path planning component in order to generate a safer trajectory for the mobile robot.

We conducted our experiments with data collected while the mobile robot shown in Figure 1a was navigating on a forest road, with dynamic obstacles present in its vicinity. Dynamic obstacles were represented by human subjects approaching the mobile robot while it navigated in the forest road environment. Our experiments indicate a better performance than the baseline.

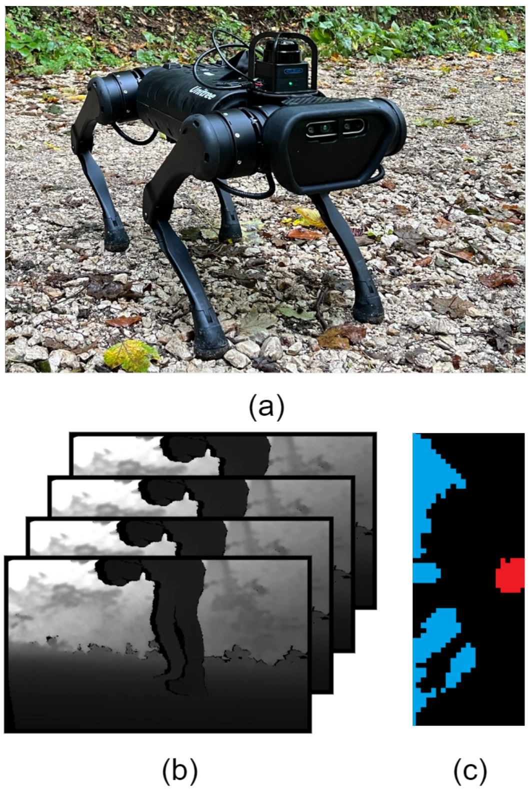

Figure 1. Vision Dynamics–Collision Detection. (**a**) A legged robot navigating in an unstructured environment. (**b**) RGB-D sequence of observations provided as input to a deep neural network. (**c**) Collision detection output as an occupancy birds-eye view grid.

The main contributions of this paper are as follows:

- The deep neural network architecture used to encode the dynamics of the robot's environment and predict future sensor observations based on previously gathered information.
- The self-supervised training of our encoder network with processed sensor information from an RGB-D camera.
- The processing of the acquired RGB-D data in the scope of reducing the search space of the planning algorithm.
- The usage of a custom asymmetrical loss function for training the neural network.

- The path planning component, which incorporates the predicted observations when generating safe trajectories for the mobile robot.

The rest of the work is organized as follows. The related work is covered in Section 2. Our proposed vision dynamics algorithm is presented in Section 3. The experimental setup, a detailed description, and comparisons with other algorithms are presented in Section 4. The conclusions are presented in the final section.

2. Related Work

In recent years, the development of autonomous navigation solutions has seen a rise in popularity in the scientific community, mainly due to the advances of artificial intelligence. Several papers which explore various aspects of the autonomous navigation task, like perception and control, have been published.

Some research treats the entire autonomous navigation problem as a black-box. The proposed algorithms expect sensory information as input to a deep learning model and directly output control signals. For example, one such approach can be found in reference [1], where the authors train a convolutional neural network to directly map pixel information from a monocular camera to vehicle steering commands. This approach tries to simplify the autonomous navigation problem by eliminating the necessity of individual sub-systems like lane marking detection or path planning. Another approach uses the Deep Reinforcement Learning (DRL) technique, where the agent is controlled via action–reward systems and is stimulated to take actions to advance toward its goal, as in [2].

Both End-to-End and DRL approaches show promising results in a controlled simulated environment; however, the deployment of such algorithms in real-world scenarios is difficult, due to the high unpredictability of the environment dynamics.

One example of Deep Reinforcement Learning being used in an unstructured navigation task is the work presented in [3]. The authors used a simulated environment to perform the training of their model, enabling the system to learn navigation policies and then perform transfer learning as *sim-to-real*.

Another example of Reinforcement Learning employed to solve the unstructured navigation problem is presented in [4]. Their proposal contains a framework based on an uncertainty-aware learning based model and a self-supervised traversability estimation model. The traversability model is then integrated with a model predictive controller, which optimizes the vehicle's trajectory while simultaneously avoiding obstacles present in the environment.

Other work relies on object detection algorithms in order to identify obstacles from the scene and to generate collision-free trajectories for the autonomous agent. Given the large number of publicly available driving databases, object detection based on image data is a widely researched area as well. In [5], the authors propose training a state-of-the-art object detector algorithm in the hope of improving performance for self-driving scenarios. Wang et al. proposed a collision detection mechanism based on a convolutional neural network in [6] within the context of autonomous driving. Pramanik et al. proposed an object detection and tracking system which handles video sequences in [7].

More recently, in [8], an obstacle avoidance method was proposed based on object detection, with certain improvements in their model, like in the variable learning rate.

Object detection algorithms have high computational requirements and necessitate various optimizations and improvements before being able to run in real-time. They also face issues such as robustness to occlusions, illumination changes, and object deformation.

Another approach to the autonomous driving challenge, particularly to the collision avoidance strategy, is based on occupancy grids, where the environment is represented as a grid of cells, each cell having an assigned probability that indicates whether or not the respective cell is occupied. Shepei et al. proposed an occupancy grid generation algorithm based on stereo image input in [9]. Their architecture is based on ResNet, trained to highlight static and dynamic obstacles within the scene.

Another type of occupancy grid generation algorithm makes use of LiDAR data instead of stereo cameras, such as the one proposed in [10], where the authors integrate this algorithm into a quadrotor system and perform benchmark flights. Usually, algorithms that handle LiDAR data suffer from high computation costs and limited scalability, as they require the processing of large amounts of data.

Recurrent neural networks have also been in the spotlight for autonomous navigation solutions. For example, the authors of [11] present a multi-step prediction of drivable space in the form of occupancy grid sequences, using an RNN architecture that was trained on the KITTI dataset, in the scope of learning motion-related features from the input sequences.

There are also approaches to the unstructured navigation problem which do not rely on artificial intelligence. One example of such a work is [12], where the authors use A^* for calculating the global trajectory for their robotic platform, combined with a local planner and a reactive planner to ensure that the robot will keep track of the computed global trajectory.

A similar approach for a legged robot without using artificial intelligence is presented in [13]. In this study, the authors propose an end-to-end path planning and navigation system which is used by the robot to navigate in its' environment. The legged robot's trajectory is generated with the use of a nonlinear optimization method.

The method proposed in [14] uses LiDAR data, coupled with an autonomous mapless navigation method based on [15] in order to perform legged robot navigation in an unstructured, urban environment. This approach combines ranging sensing with Deep Reinforment Learning and classical path planning in order for the robot to successfully avoid the dynamic obstacles present in an urban environment.

These works demonstrate the potential of using object detection and deep-learning-based approaches for obstacle avoidance in dynamic environments. However, they still face challenges such as robustness to occlusions, illumination changes, and object deformation. The training process for these object detectors is label-intensive as well. Additionally, some of the methods use expensive sensors (e.g., LiDAR) to map the mobile robot's environment and avoid collisions with obstacles.

Basing our work on [16–18], we propose a vision dynamics approach for solving the unstructured navigation problem for a legged mobile robot, based on processing information coming from an RGB-D sensor through the use of a recurrent neural network which predicts future sensory observations and feeds these observations into a modified DWA algorithm in order to produce an obstacle-free trajectory for the mobile robot. We demonstrate the effectiveness of the approach in a real-world scenario during a navigation task on a forest road.

3. Methodology

The following notation is used throughout this paper. Superscript $<t>$ is used to denote a single discrete time step t. For example, a depth sensor measurement at time step t will be written as $\theta^{<t>}$. Predicted values are defined with the hat notation. A predicted observation at time step $t+n$ is defined as $\hat{\theta}^{<t+n>}$. Vectors are represented by bold symbols. A vector of sensor observations is marked as $\mathbf{\Theta}^{<t-\tau,t-1>}$, which represents a series of sensor observations from time step $t-\tau$ to time step $t-1$.

3.1. Problem Definition

Given a set of historical depth sensing observations $\Theta^{<t-\tau,t-1>}$, the current sensor observation $\theta^{<t>}$, the mobile robot's current state $s^{<t>} = [x^{<t>}, y^{<t>}, v^{<t>}, \phi^{<t>}]$, and a global reference trajectory $\mathbf{Z}_{ref}^{<t-\infty,t+\infty>}$, the objective is to construct a recurrent deep neural network model which can encode within its layers the dynamics of the environment and, based on historical sensor observations, can predict future observations for a finite time horizon $[t+1,\ldots,t+n]$, where each time step $t+1, t+2, \ldots$ represents one future sampling time, bound by a time constraint of 100 ms. The agent's belief, modelled with the neural network, is passed to the path planner algorithm based on the Dynamic Window Approach and a collision-free local trajectory is generated for the mobile robot: $\mathbf{Z}_L^{<t+1,t+n>}$.

Both local and global trajectories are represented as sequences of vehicle states: $\mathbf{Z}_{ref} = \{[x_i, y_i, v_i, \phi_i] | \forall i \in [1, \dots, N]\}$ and, respectively, $\mathbf{Z}_L = \{[x_i, y_i, v_i, \phi_i] | \forall i \in [1, n]\}$, where x_i and y_i are the Cartesian coordinates of the vehicle, v_i the velocity, and ϕ_i is the orientation angle of the vehicle with respect to its initial orientation. n is the length of the generated local trajectory, $\mathbf{Z}_{ref}$ is the global reference trajectory, and $\mathbf{Z}_L$ is the generated local trajectory.

3.2. Scene Dynamics Encoder Network

The architecture of the encoder model is presented in Figure 2. The observations coming from the RGB-D sensor are not directly processed by the network, but undergo a series of preprocessing operations beforehand, as described in Equations (2)–(4). The input sequence (historic observations) is accumulating in the *Encoder memory unit*, formatted appropriately by the *Sequence formatter* block, and then passed through to the recurrent neural network. The *Clustering unit* clusters the predictions coming from the neural network and eliminates outliers. Encoding scene dynamics involves processing data coming from the RGB-D sensor in order to create a representation of the current scene, as well as predicting the evolution of the scene over a fixed time period. LSTM networks are well suited for encoding scene dynamics due to their ability to learn the temporal dependencies of the input data and use these dependencies for predicting future states.

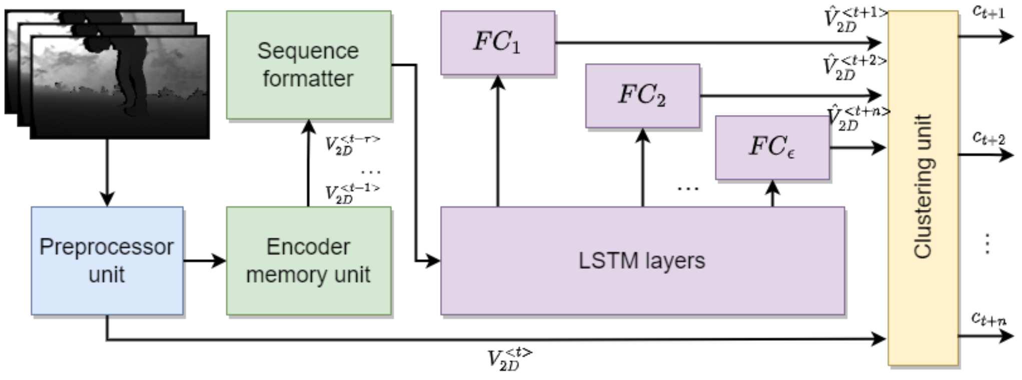

Figure 2. Encoder model architecture. The input of the deep neural network is a sequence of length τ observations coming from the RGB-D sensor.

The encoder model is based on an LSTM architecture which contains multiple cells for processing time-series data in the form of historical depth sensor observations of length τ. The output of the model is a sequence of future observations of length n.

Depth information coming from the RGB-D sensor is not fed directly into the network, but it is processed beforehand.

First, using depth measurements from the sensor and, respectively, the depth camera intrinsic matrix, the three-dimensional environment is recreated:

$$\mathbf{P}_{3D} = \{[X_i, Y_i, Z_i] | \forall i \in \mathbf{D}_{img}\} \tag{1}$$

where X_i, Y_i, and Z_i are the three-dimensional coordinates of the points P; D_{img} is the depth image. For every point in the depth image domain, corresponding $3D$ points are calculated:

$$\begin{aligned} X &= (j - c_x)\frac{z}{f_x} \\ Y &= (i - c_y)\frac{z}{f_y} \\ Z &= z \end{aligned} \tag{2}$$

where j is the column coordinate of the current pixel from the depth image, i is the row coordinate of the current pixel from the depth image, c_x and c_y are the optical center coordinates, f_x and f_y are the focal distance coordinates, and z is the depth value at coordinates (i, j).

Second, only candidate points that are closer than a threshold value are kept:

$$p \Leftarrow p_c \iff T_{min} \leq d(o, p_c) \leq T_{max}, \forall p_c \in P_{3D} \tag{3}$$

where p is the filtered point, p_c is the candidate point, o is the vehicle coordinate system origin, T_{min} is the minimum threshold value, T_{max} is the maximum threshold value, and d is the L2 norm, calculated as $||o - p_c||^2$.

Third, points that are below or above a certain height are eliminated:

$$p \Leftarrow p_c \iff T_{min} \leq Y_{p_c} \leq T_{max}, \forall p_c \in P_{3D} \tag{4}$$

where p is the filtered point, p_c is the candidate point, T_{min} is the minimum threshold value, T_{max} is the maximum threshold value, and Y_{p_c} is the height of the candidate point p_c. Eliminating points below the minimum threshold will also eliminate points that belong to the ground, therefore easing further computation steps.

The resulting point cloud is then converted into a voxel representation in order to reduce the input data dimension:

$$\mathbf{V_{3D}} = \begin{cases} 1 & , if P_i \in v(i, j, k) \\ 0 & , otherwise \end{cases} \tag{5}$$

where $v(i, j, k)$ is a 3D space within the voxel grid.

Furthermore, the voxel representation is then projected onto a top-view occupancy grid:

$$\mathbf{V_{2D}} = \{(X_i, Y_i) | X_i \in V_{3D}, Y_i \in V_{3D}\} \tag{6}$$

These two-dimensional grids accumulate into a historical sequence that is then fed as input to the sequence formatter module from our algorithm.

$$\mathbf{V_{2D}}^{<t-\tau,t-1>} \Leftarrow \{V_{2D}^{<i>}, \forall i \in [t-\tau, t-1]\} \tag{7}$$

A visual representation of the processing chain applied to the input depth images can be seen in Figure 3. The history sequences $\mathbf{V}_{2D}^{<t-\tau,t-1>}$ are, again, processed by the sequence formatter module, which reshapes them to a suitable format for the LSTM module of our algorithm.

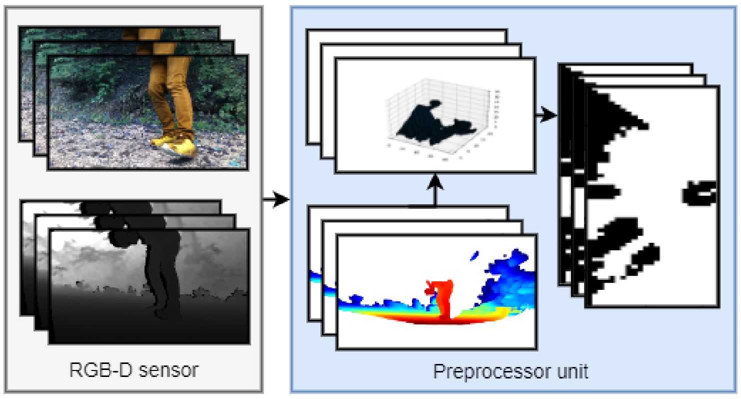

Figure 3. Preprocessing chain. Sensor data coming from the RGB-D camera is processed before being stored into the *Encoder memory unit*. Depth information is reconstructed using the camera intrinsic parameters into a three-dimensional point cloud. The point cloud is then converted to a voxel grid representation. The three-dimensional voxel grid is then projected into a top-down view.

At the heart of the scene dynamics encoder model there is a stacked LSTM module with n layers. The output of each layer is then fed through a series of fully connected layers in order to produce the predictions at time steps $[t+1,\dots,t+n]$.

The resulting predictions $\hat{\mathbf{V}}_{\mathbf{2D}}^{<t+1,t+n>}$ coming from the neural network, in the form of predicted voxel grids for future time steps $[t+1,\dots,t+n]$, are passed to the *clustering unit* for further processing. The clustering unit takes each predicted observation $V_{2D}^{<t+i>}$ and performs DBSCAN, according to [19]. The output of this layer is, for each time step $t+i$, a list of centroids plus the object size corresponding to each obstacle present in the current time step. Obstacle points are sampled based on the centroids and sizes provided by the clustering unit for each predicted time step $t+i$ and are accumulated in order to be fed to the DWA algorithm.

The clustering unit also acts as a filter mechanism for the output data, using the *Log Odds* method.

The data processing main workflow is presented in Algorithm 1.

Algorithm 1 Processing loop

```
Require: $\mathbf{\Theta}^{<t-\tau,t-1>}$
Require: $\theta^{<t>}$                                  ▷ Current depth sensor observation
Require: $obstacles = \{\}$                              ▷ Empty list of obstacles
 1: for $i \in [t-\tau, t]$ do
 2:     $P_{3D}^{<i>} \leftarrow$ to_pcl($\theta^{<i>}$)
 3:     $P_{3D}^{<i>} \leftarrow$ reject_outliers($P_{3D}^{<i>}$)
 4:     $V_{3D}^{<i>} \leftarrow$ to_3d_voxel($P_{3D}^{<i>}$)
 5:     $V_{2D}^{<i>} \leftarrow$ project_2d($V_{3D}^{<i>}$)
 6: end for
 7: $m \leftarrow \{V_{2D}^{<i>} | \forall i \in [t-\tau, t-1]\}$
 8: $\hat{\mathbf{V}}_{2D}^{<t+1,t+n>} \leftarrow m(\mathbf{V}_{2D}^{<t-\tau,t-1>})$          ▷ Obtain predictions
 9: $\mathbf{c}_j \leftarrow$ cluster_outputs($\hat{v}_{2D}^{<t+1,t+n>}, V_{2D}^{<t>}$)        ▷ Cluster outputs
10: for $j \in [0, c_{len}]$ do
11:     $obstacles \leftarrow c_j$                           ▷ Append obstacle
12: end for
```

The RNN has been trained in a self-supervised way, over a number of 1000 epochs, using an Adam optimizer with a learning rate of 0.001. The history size used for training was $\tau = 50$ and the prediction horizon size was $n = 30$. This is equivalent to 5 s for the history sequence and 3 s for prediction, respectively. For training the network, we used a total number of 30 K pairs of input historical depth observations and target predictions.

A custom asymmetrical loss function was used, as shown in Equation (8), because of the sparse nature of the output sequence data.

$$L(V,\hat{V}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{V}_i - V_i)^2[(1-1_+)\alpha + 1_+(1-\alpha)] \tag{8}$$

where α is a scale factor that increases the loss value for predictions that diverge from target values and 1_+ is a function defined as follows:

$$1_+(V,\hat{V}) = \begin{cases} 1 & ,\hat{V} - V \geq 0 \\ 0 & ,otherwise \end{cases} \tag{9}$$

After experimenting with the architecture and the loss function, the scale factor α was set to 0.1.

A visual representation of the asymmetrical loss function is presented in Figure 4. It can be observed that the function penalizes the model if the predicted voxels are marked as occupied, when, instead, they should be marked as free-space.

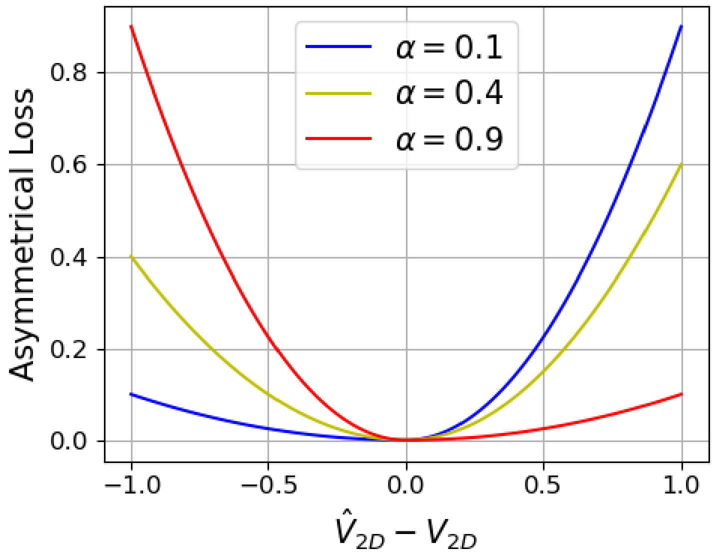

Figure 4. Asymmetrical Loss function. The asymmetrical loss function calculated for different values of the scale factor α.

3.3. Path Planning Algorithm

For controlling the mobile robot, we used the Zero Moment Point (ZMP) algorithm [20], which is described briefly in the following:

The Zero Moment Point can be described as the point $\vec{z}$ on the ground where the net moment along the vertical axis passing through $\vec{z}$ is zero:

$$\sum_i (\vec{r}_i - \vec{z}) \times \vec{f}_i = 0 \tag{10}$$

where $\vec{r}_i$ is the position of the i-th foot contact point relative to $\vec{z}$ and $\vec{f}_i$ is the ground reaction force at the i-th foot contact point

Assuming the center of mass does not move, the ZMP can be approximated as follows:

$$\vec{z} = \vec{p} - \frac{1}{mg} \sum_i \vec{f}_i \times \vec{r}_i \tag{11}$$

where m is the mass of the robot, g is the gravitational acceleration, and $\vec{p}$ is the position of the robot's center of mass.

The desired target ZMP $\vec{z}_d$ is calculated based on the desired motion (e.g., walking forward):

$$\vec{z}_d = f(\vec{p}_d, \vec{f}_1, \ldots, \vec{f}_n) \tag{12}$$

where f is a function that maps the desired motion to the target ZMP. The robot motion is then adjusted to bring the actual ZMP close to the target ZMP. The desired ZMP $\vec{z}_d$ was chosen based on the output of the Dynamic Window Approach algorithm, by selecting the destination point from the optimal generated trajectory. Our DWA implementation is based on a simplified non-holonomic robot model commonly encoutered in wheeled locomotion:

$$\begin{cases} x_{t+1} = x_t + v_{t+1} \cos \Theta \\ y_{t+1} = y_t + v_{t+1} \sin \Theta \\ \Theta_{t+1} = \Theta_t + \delta_{t+1} \end{cases} \tag{13}$$

where x and y represent the robot's position in the 2D birds-eye view moving space, Θ is the heading, and δ the desired direction.

DWA first calculates a dynamic window w for the commands of the robot, based on physical constraints regarding velocity and acceleration. Each command v_{t+1}, δ_{t+1} from the generated dynamic window is used to produce a candidate trajectory. Afterwards, predictions coming from the scene dynamics encoder network are used, together with current sensor observations to reject candidate trajectories that intersect any obstacle (predicted or observed). Each candidate trajectory that was not rejected in the previous step is then analyzed with respect to several performance indicators, and the following penalties are computed (see Algorithm 2):

- Goal cost (ϵ_G), which represents the distance between the Cartesian coordinates of the final state on the candidate trajectory and the coordinates of the goal point located on the global reference trajectory.
- Orientation cost (ϵ_ϕ), which represents the difference between the orientation angle from the final state of the candidate trajectory and the orientation angle from the reference orientation angle.
- Velocity cost (ϵ_v), which represents the difference between the mobile robot's velocity and the reference velocity.
- Smoothness cost (ϵ_s), which represents the difference between the previous command and the current command (if there is a significant difference between successive commands, then the motion of the mobile robot will become course).
- Trajectory cost (ϵ_t), which represents the sum of distances between Cartesian coordinates of the candidate trajectory and the reference trajectory.

Algorithm 2 DWA control loop

```
Require: $\mathbf{Z}_{ref}^{<t-\infty,t+\infty>}$                ▷ Global trajectory
Require: $\hat{\mathbf{V}}_{2D}^{<t+1,t+n>}$                 ▷ Predicted observations
Require: $V_{2D}^{<t>}$                          ▷ Current observation
Require: $s^{<t>}$                             ▷ Current robot state
Require: $ZMP \leftarrow []$
 1: $w \leftarrow$ dwa_window()                  ▷ Calculate dynamic window
 2: $min_C = \infty$
 3: for $v, \delta \in w$ do
 4:     candidate $\leftarrow$ dwa_loop($v, \delta$)
 5:     candidate $\leftarrow$ reject_collision(candidate, $s^{<t>}$, $\hat{\mathbf{V}}_{2D}$)
 6:     if candidate $\neq \emptyset$ then
 7:         $\epsilon_G \leftarrow$ goal_cost($v, \delta, \mathbf{Z}_{ref}$)
 8:         $\epsilon_\phi \leftarrow$ orientation_cost($v, \delta, \mathbf{Z}_{ref}$)
 9:         $\epsilon_v \leftarrow$ velocity_cost($v, \delta, \mathbf{Z}_{ref}$)
10:         $\epsilon_s \leftarrow$ smoothness_cost($v, \delta, v^{<t-1>}$,
11:         $\delta^{<t-1>}$)
12:         $\epsilon_t \leftarrow$ trajectory_cost($v, \delta, \mathbf{Z}_{ref}$)
13:         $C = K_G\epsilon_G + K_\phi\epsilon_\phi + K_v\epsilon_v + K_s\epsilon_s + K_t\epsilon_t$
14:         if $C < min_C$ then
15:             $min_C \leftarrow C$
16:             $ZMP \leftarrow candidate[n]$
17:         end if
18:     end if
19: end for
```

The collision rejection method parses all the centroids returned by the scene dynamics encoder module and rejects all candidate trajectories which intersect obstacles, either seen in the current timestamp observations or predicted by the neural network. After evaluating the total cost C for each of the candidate trajectories, the one with the minimum cost is selected and the ZMP gets updated.

4. Experiments

For evaluating our algorithm, we used sequences of data recorded during the navigation process of our mobile robot in the unstructured environment, more specifically, a forest road navigation task. The mobile robot shown in Figure 1a navigated within the environment, while potential collision situations were generated. In the meantime, depth sensor data were captured and analyzed during navigation. A global reference trajectory $\mathbf{Z}_{ref}^{<t-\infty,t+\infty>}$ was recorded for the mobile robot.

Samples of the data processed by the algorithm can be viewed in Figure 5. Depth images—Figure 5b—are used coupled with the camera parameters in order to obtain a 3D representation of the scene—Figure 5c. Then, 3D points are processed according to Section 3.2 and the three-dimensional voxel grid is computed—Figure 5d. This voxel grid is afterwards projected in a top-down view—Figure 5e. RGB and depth images were collected while the legged robot navigated on forest roads and interacted with humans, who acted as obstacles for our algorithm. In Figure 5a, a multiple-obstacle scenario is presented; the two persons are approaching the legged robot while it records RGB and depth images. The thresholding performed on the reconstructed three-dimensional points can be observed in Figure 5c. Ground points, as well as points that are too high or too far away from the robot, are eliminated from the resulting point cloud before the 3D voxel grid is computed.

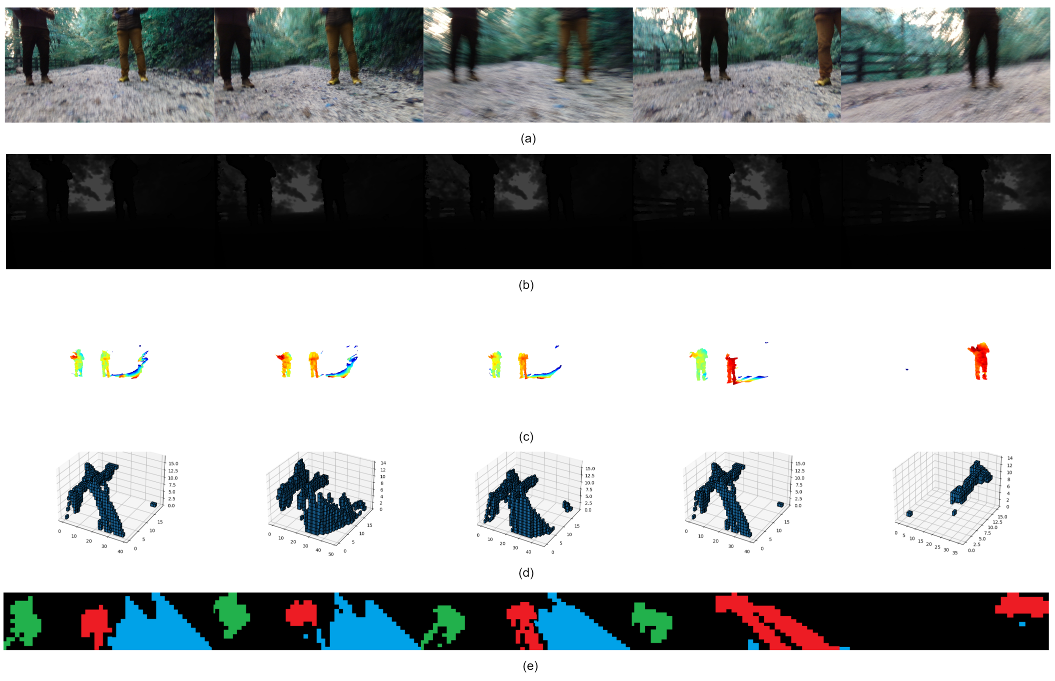

Figure 5. Experimental data processing chain. The experimental data used by the algorithm. From top to bottom: (**a**)—RGB data, (**b**)—depth image, (**c**)—3D representation, (**d**)—3D voxel grid, and (**e**)—2D voxel grid.

The experiments were structured as follows:

- Training and testing data were acquired from the navigation scenario.
- The scene dynamics encoder network was trained using the preprocessed training data.
- The path planning algorithm was evaluated on the preprocessed test data.

We defined a safe distance $s = 0.5$ m for our mobile robot. Each violation of this distance, which meant that the mobile robot approached any obstacle within a radius less than s, automatically triggered a *collision event*. Collision events were counted, producing the first quality measure: n_{col}.

The second evaluation metric was the cross-track error, defined as the difference between the reference trajectory Z_{ref} and the generated local trajectory:

$$\bar{e}_{ct} = \frac{1}{n}\sum_{i=0}^{n} |f(x) - y_i| \tag{14}$$

where $f(x)$ is the polynomial approximation of the generated local trajectory evaluated in x and y_i is the measured point y coordinate.

The third evaluation metric was the orientation error:

$$\bar{e}_{\phi} = \frac{1}{n}\sum_{i=0}^{n} |\phi^{<i>} - \phi_d^{<i>}| \tag{15}$$

where $\phi^{<i>}$ is the robot's orientation at time step i and $\phi_d^{<i>}$ is the desired orientation from the global reference trajectory.

In order to reproduce potentially hazardous situations for the mobile robot, we experimented with people interacting with it during navigation. Interactions consisted of people approaching the robot from different angles and with varying velocity, in order to simulate dynamic obstacles.

We simulated four scenarios: front collision, side left, side right, and multiple obstacles. For the first scenario, a single person approached the mobile robot head on, while for the side collision left and side collision right scenarios, the human would approach the robot from the left and from the right sides, respectively. For the multiple obstacles setup, we considered two persons for our experiments.

We compared our approach to the classic Dynamic Window Approach path planner, configured with the same dynamic window generation parameters, as described in the Section 3.

Additionally, we strictly compared our scene dynamics encoder network with a state-of-the-art object detection algorithm, YoloV7 trained on the MS COCO dataset, as in the original implementation from [21], in terms of accuracy, false positive rate—FP_R and time to collision—ttc on our test data. The MS COCO dataset contains annotations for object detection (bounding boxes), semantic and instance segmentation, etc. [22] The time to collision is the estimated time elapsed between the first available detection of an obstacle until the respective obstacle crosses the safe distance threshold.

The results of the comparison between path planning algorithms are summarized in Table 1, while the results of the comparison between neural networks are presented in Table 2.

It can be observed that our proposal surpasses the baseline in all four test scenarios, both in the first experiment, as a path planning component, and in the second one, as a means of detecting dynamic obstacles present in the scene.

Table 1. Qualitative results on real-world mobile robot setup for path planning algorithm.

Scenario	**Algorithm**	n_{col}	$\bar{e}_{ct}(m)$	$\bar{e}_{\phi}(rad)$
Front	Ours	**7**	**0.35**	**0.12**
	DWA	27	0.43	0.15
Left	Ours	**5**	**0.37**	0.12
	DWA	33	0.43	0.12
Right	Ours	**3**	**0.29**	**0.1**
	DWA	32	0.39	0.11
Multiple	Ours	**5**	0.32	0.16
	DWA	37	0.32	**0.12**

Table 2. Qualitative results on real-world mobile robot setup for deep learning architectures.

Scenario	Algorithm	$t\bar{t}c(s)$	$FP_R(\%)$	$A(\%)$
Front	Ours	**2.5**	**10**	**95**
	YoloV7	2.3	12	92
Left	Ours	**1.2**	**10**	**93**
	YoloV7	1.3	11	92
Right	Ours	**1.3**	11	92
	YoloV7	1.0	11	92
Multiple	Ours	**1.7**	14	**88**
	YoloV7	1.3	**13**	82

The number of collisions n_c was significantly lower in all four testing scenarios in our approach than in the case of classical DWA. This suggests that including predicted observations in the trajectory generation step improves the obstacle avoidance capability of the system. The cross-track error $\bar{e}_{ct}$ and orientation error $\bar{e}_{\phi}$, respectively, were predominantly lower using our proposed algorithm than in the case of classical DWA. This is caused by the experimental tuning of the parameters included in our version of DWA (trajectory cost ϵ_t and orientation cost ϵ_{ϕ}).

The time to collision (ttc), false positive rate (FP_R), and accuracy (A) values were better in our approach than the YoloV7 variant because, in our proposal, the algorithm had more data to decide upon, due to the inclusion of predictions for future voxel grids. Additionally, using depth information instead of image data benefits the algorithm due to the fact that there is no need to perform error-prone object recognition.

Ablation Study

For the purpose of assessing the performance of the components from our algorithm, we performed three ablation studies.

In the first one, we compared three loss functions for our neural network: Mean Squared Error, Mean Absolute Error, and Asymmetrical Loss, respectively. It can be observed that the Asymmetrical Loss function converges the fastest out of the three functions, while the loss value is the lowest. We compared the three functions over 500 epochs for data in the validation set. The results can be viewed in Figure 6.

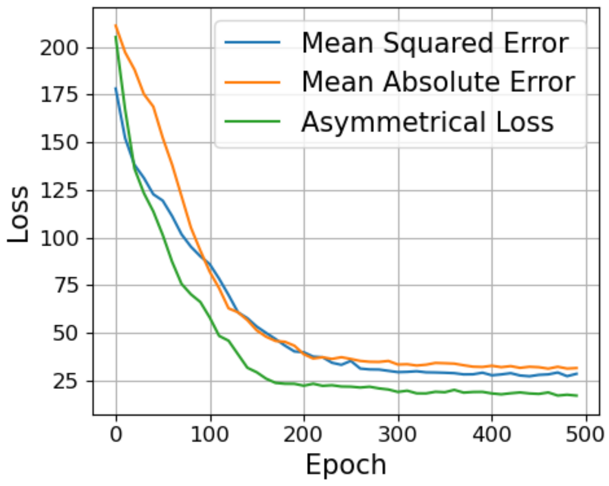

Figure 6. Loss functions ablation. Ablation of Mean Squared Error, Mean Absolute Error and Asymmetrical Loss functions. The Asymmetrical Loss function produces the lowest loss and converges the fastest out of the three functions.

The second study concerns the ablation of the prediction horizon size versus the time to collision estimate (ttc) and false positive rate FP_R. It is obvious that, by increasing the prediction horizon size, the ttc values improve slightly; however, this also produces a higher FP_R. Our experiments suggest that the increase of the prediction horizon size generates more noise in the predicted observations, which implicitly causes the algorithm to signal more collisions. The results of the prediction horizon size ablation are shown in Figure 7.

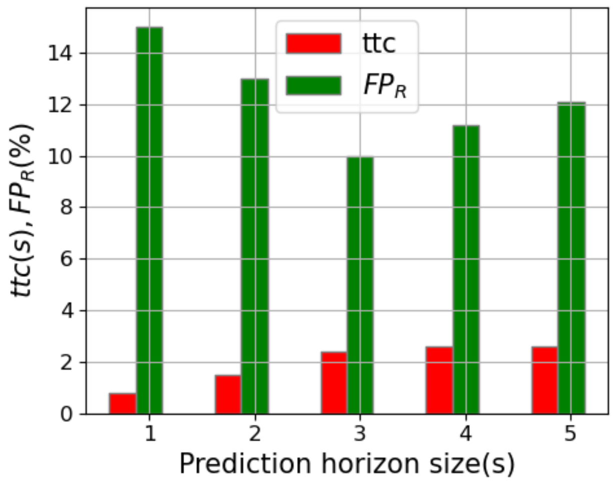

Figure 7. Prediction horizon size ablation. Prediction horizon size vs. performance metrics: ttc and FP_R. A slight improvement of the time to collision metric is observed when the algorithm predicts observations for a longer prediction horizon.

For the third ablation study, we compared the performance of the algorithm for different values of the history size parameter. For the comparison, we used the following values (in seconds): $[1, 3, 5, 7, 10]$. It can be observed that increasing the history size to a value larger than 5 s does not bring significant improvements to our algorithm. There is a slight improvement in the test result; however, the disadvantages outweigh the benefits. The results of the history size ablation are presented in Figure 8.

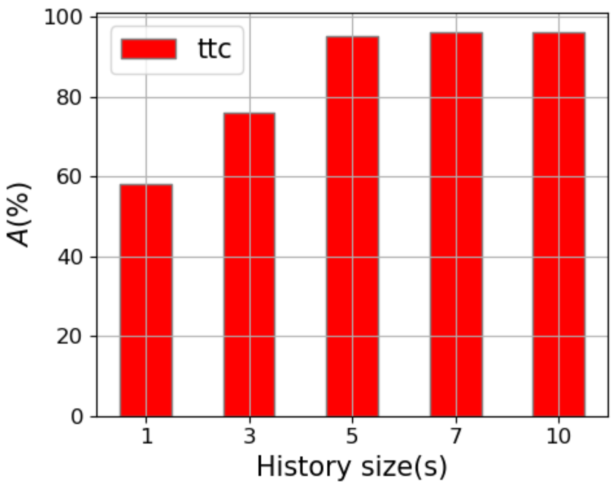

Figure 8. History size ablation. History size vs. accuracy performance metric. A slight improvement can be seen when presenting a longer sequence of historical data.

5. Conclusions

In this paper, we have introduced a vision dynamics learning approach for mobile robot navigation in unstructured environments. Our model is based on a recurrent neural network that processes historical data coming from a depth sensor and produces predictions for future time steps over a fixed prediction horizon. We also described the process of converting the depth image into a three-dimensional point cloud, and then voxelizing the point cloud to a bidimensional grid of voxels that can serve as input for our deep learning model.

We have evaluated our model on real-world data, which was gathered during the mobile robot's navigation on a forest road. The ability to directly encode scene dynamics using a deep learning model has significant implications for robotics, as it can enable a more accurate understanding of the surroundings and more precise interactions between the robot and its environment, thus contributing to better decision making in tasks such as autonomous navigation.

Future work can explore using more complex neural network architectures, as well as integrating additional sensor inputs to the deep learning model, such as images or LiDAR data, in order to further improve the accuracy and robustness of the model.

Author Contributions: Conceptualization, L.F. and S.G.; methodology, C.G. and M.Z.; software, C.G.; validation, M.Z., L.F., D.C. and S.G.; formal analysis, C.G., D.C. and S.G.; investigation, C.G. and M.Z.; resources, L.F.; writing—original draft, C.G.; writing—review and editing, M.Z. and D.C.; supervision, S.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Romanian Executive Agency for Higher Education, Research, Development, and Innovation Funding under Grant no. PED675/2022.

Data Availability Statement: Data are contained within the article.

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

References

1. Bojarski, M.; Testa, D.D.; Dworakowski, D.; Firner, B.; Flepp, B.; Goyal, P.; Jackel, L.D.; Monfort, M.; Muller, U.; Zhang, J.; et al. End to End Learning for Self-Driving Cars. *arXiv* **2016**, arXiv:1604.07316.
2. Jaritz, M.; de Charette, R.; Toromanoff, M.; Perot, E.; Nashashibi, F. End-to-End Race Driving with Deep Reinforcement Learning. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 2070–2075. [CrossRef]
3. Truong, J.; Yarats, D.; Li, T.; Meier, F.; Chernova, S.; Batra, D.; Rai, A. Learning Navigation Skills for Legged Robots with Learned Robot Embeddings. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; pp. 484–491. [CrossRef]
4. Seo, J.; Mun, J.; Kim, T. Safe Navigation in Unstructured Environments by Minimizing Uncertainty in Control and Perception. *arXiv* **2023**, arXiv:2306.14601.
5. Li, G.; Ji, Z.; Qu, X.; Zhou, R.; Cao, D. Cross-Domain Object Detection for Autonomous Driving: A Stepwise Domain Adaptative YOLO Approach. *IEEE Trans. Intell. Veh.* **2022**, *7*, 603–615. [CrossRef]
6. Wang, X.; Liu, J.; Qiu, T.; Mu, C.; Chen, C.; Zhou, P. A Real-Time Collision Prediction Mechanism With Deep Learning for Intelligent Transportation System. *IEEE Trans. Veh. Technol.* **2020**, *69*, 9497–9508. [CrossRef]
7. Pramanik, A.; Pal, S.K.; Maiti, J.; Mitra, P. Granulated RCNN and Multi-Class Deep SORT for Multi-Object Detection and Tracking. *IEEE Trans. Emerg. Top. Comput. Intell.* **2022**, *6*, 171–181. [CrossRef]
8. Rezaei, N.; Darabi, S. Mobile robot monocular vision-based obstacle avoidance algorithm using a deep neural network. *Evol. Intell.* **2023**, *16*, 1999–2014. [CrossRef]
9. Shepel, I.; Adeshkin, V.; Belkin, I.; Yudin, D.A. Occupancy Grid Generation With Dynamic Obstacle Segmentation in Stereo Images. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 14779–14789. [CrossRef]
10. Ren, Y.; Cai, Y.; Zhu, F.; Liang, S.; Zhang, F. ROG-Map: An Efficient Robocentric Occupancy Grid Map for Large-scene and High-resolution LiDAR-based Motion Planning. *arXiv* **2023**, arXiv:2302.14819.
11. Mohajerin, N.; Rohani, M. Multi-Step Prediction of Occupancy Grid Maps With Recurrent Neural Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 10592–10600. [CrossRef]
12. Li, Z.; Zeng, J.; Chen, S.; Sreenath, K. Autonomous navigation of underactuated bipedal robots in height-constrained environments. *Int. J. Robot. Res.* **2023**, *42*, 565–585. [CrossRef]
13. Gilroy, S.; Lau, D.; Yang, L.; Izaguirre, E.; Biermayer, K.; Xiao, A.; Sun, M.; Agrawal, A.; Zeng, J.; Li, Z.; et al. Autonomous Navigation for Quadrupedal Robots with Optimized Jumping through Constrained Obstacles. In Proceedings of the 2021 IEEE 17th International Conference on Automation Science and Engineering (CASE), Lyon, France, 23–27 August 2021; pp. 2132–2139. [CrossRef]

14. Fan, T.; Chen, Z.; Zhao, X.; Liang, J.; Shen, C.; Manocha, D.; Pan, J.; Zhang, W. Autonomous Social Distancing in Urban Environments using a Quadruped Robot. *IEEE Access* **2021**, *9*, 8392–8403.
15. Fan, T.; Cheng, X.; Pan, J.; Manocha, D.; Yang, R. CrowdMove: Autonomous Mapless Navigation in Crowded Scenarios. *arXiv* **2018**, arXiv:1807.07870.
16. Grigorescu, S.; Ginerica, C.; Zaha, M.; Macesanu, G.; Trasnea, B. LVD-NMPC: A learning-based vision dynamics approach to nonlinear model predictive control for autonomous vehicles. *Int. J. Adv. Robot. Syst.* **2021**, *18*, 17298814211019544. [CrossRef]
17. Trăsnea, B.; Ginerică, C.; Zaha, M.; Măceşanu, G.; Pozna, C.; Grigorescu, S. OctoPath: An OcTree-Based Self-Supervised Learning Approach to Local Trajectory Planning for Mobile Robots. *Sensors* **2021**, *21*, 3606. [CrossRef] [PubMed]
18. Ginerica, C.; Zaha, M.; Gogianu, F.; Busoniu, L.; Trasnea, B.; Grigorescu, S. ObserveNet Control: A Vision-Dynamics Learning Approach to Predictive Control in Autonomous Vehicles. *IEEE Robot. Autom. Lett.* **2021**, *6*, 6915–6922. [CrossRef]
19. Ester, M.; Kriegel, H.P.; Sander, J.; Xu, X. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*; In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, OR, USA, 2–4 August 1996; AAAI Press: Portland, OR, USA, 1996.
20. Akbas, T.; Eskimez, S.E.; Ozel, S.; Adak, O.K.; Fidan, K.C.; Erbatur, K. Zero Moment Point based pace reference generation for quadruped robots via preview control. In Proceedings of the 2012 12th IEEE International Workshop on Advanced Motion Control (AMC), Sarajevo, Bosnia and Herzegovina, 25–27 March 2012; pp. 1–7. [CrossRef]
21. Wang, C.Y.; Bochkovskiy, A.; Liao, H.Y.M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv* **2022**, arXiv:2207.02696.
22. Lin, T.Y.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramanan, D.; Zitnick, C.L.; Dollar, P. Microsoft COCO: Common Objects in Context. In *Computer Vision—ECCV*; Springer: Cham, Switzerland, 2014; Volume 8693, pp. 740–755. [CrossRef]