Article

OctoPath: An OcTree-Based Self-Supervised Learning Approach to Local Trajectory Planning for Mobile Robots

Bogdan Trăsnea [1,2,*], Cosmin Ginerică [1,2], Mihai Zaha [1,2], Gigel Măceşanu [1,2], Claudiu Pozna [1] and Sorin Grigorescu [1,2]

1   Robotics, Vision and Control Laboratory (ROVIS), Transilvania University of Brasov, 500036 Brasov, Romania; cosmin.ginerica@unitbv.ro (C.G.); mihai.zaha@unitbv.ro (M.Z.); gigel.macesanu@unitbv.ro (G.M.); cp@unitbv.ro (C.P.); s.grigorescu@unitbv.ro (S.G.)
2   Elektrobit Automotive, 500365 Brasov, Romania
*   Correspondence: bogdan.trasnea@unitbv.ro

Abstract: Autonomous mobile robots are usually faced with challenging situations when driving in complex environments. Namely, they have to recognize the static and dynamic obstacles, plan the driving path and execute their motion. For addressing the issue of perception and path planning, in this paper, we introduce OctoPath, which is an encoder-decoder deep neural network, trained in a self-supervised manner to predict the local optimal trajectory for the ego-vehicle. Using the discretization provided by a 3D octree environment model, our approach reformulates trajectory prediction as a classification problem with a configurable resolution. During training, OctoPath minimizes the error between the predicted and the manually driven trajectories in a given training dataset. This allows us to avoid the pitfall of regression-based trajectory estimation, in which there is an infinite state space for the output trajectory points. Environment sensing is performed using a 40-channel mechanical LiDAR sensor, fused with an inertial measurement unit and wheels odometry for state estimation. The experiments are performed both in simulation and real-life, using our own developed GridSim simulator and RovisLab's Autonomous Mobile Test Unit platform. We evaluate the predictions of OctoPath in different driving scenarios, both indoor and outdoor, while benchmarking our system against a baseline hybrid A-Star algorithm and a regression-based supervised learning method, as well as against a CNN learning-based optimal path planning method.

Keywords: sensor fusion; mobile robot systems; path planning; autonomous vehicles; artificial intelligence; deep learning

1. Introduction

Recent developments in the fields of deep learning and artificial intelligence have aided the autonomous driving domain's rapid advancement. Autonomous vehicles (AVs) are robotic systems that can navigate without the need for human intervention. The deployment of AVs is predicted to have a major impact on the future of mobility, bringing a variety of benefits to daily life, such as making driving simpler, increasing road network capacity, and minimizing vehicle-related crashes.

For Advanced Driver Assistance Systems(ADAS) systems and autonomous robot control, one of the top priorities is ensuring functional safety. When a car is driving, it encounters varieties of dynamic traffic scenarios in which the moving objects in the environment may pose a risk to safe driving. The car must consider all threats present in the surrounding environment in order to create a collision-free path and determine the next steps based on it. Due to the complexity of such a task, deep learning models have been used to aid in solving it. There are several conceptually different self-driving architectures, namely end2end learning [1], Deep Reinforcement Learning [2] (DRL), and the sense-plan-act pipeline [3].

In the sense-plan-act case, an important component is the one that plans the future driving path of the ego-vehicle. Determining a safe path over a finite prediction horizon is a key aspect when employing a control strategy, especially when considering dynamic and static obstacle avoidance. The main problem is split into smaller sub-problems in a modular pipeline, with each module intended to solve a particular task and provide the result as input to the next component. Therefore, AVs must have the ability to sense their surroundings and form an adequate environment model which precisely represents the dynamic and stationary objects. Afterward, it needs to plan its path, which is defined as the AV's ability to find a collision-free route between the current position and the desired destination. Finally, it needs to act based on the computed path, by applying the appropriate control signals (acceleration and steering) for the AV.

In this paper, we address the path planning component from the perception-planning-action pipeline. Figure 1 highlights the block diagram of the proposed concept. The currently proposed method, coined *OctoPath*, is self-supervised and aims to combine the configurable discretization of an octree-based environment model with a classification-based encoder-decoder RNN architecture. It takes as input a sequence of sensor measurements, together with the current segment of a reference trajectory, building upon the RNN encoder-decoder architecture which has shown excellent performance for sequence-to-sequence tasks.
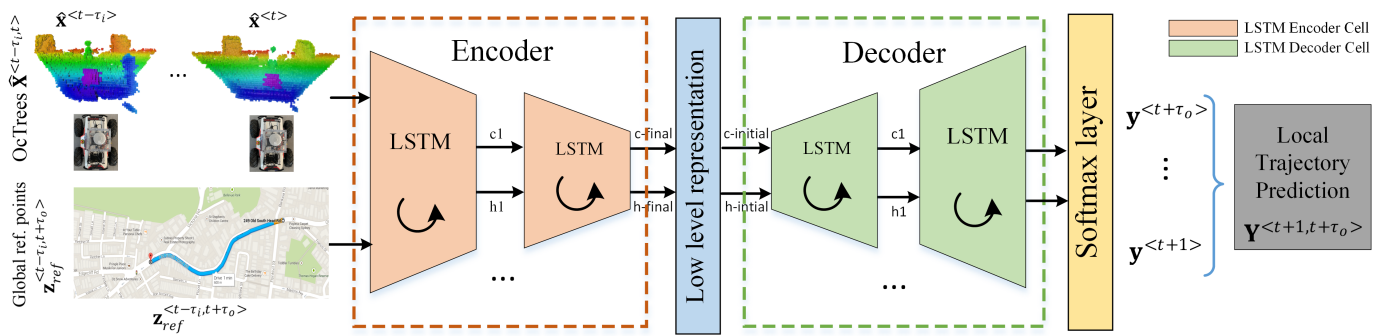

Figure 1. Local trajectory prediction using a neural network encoder-decoder architecture. The training data consists of sequences of OcTrees ($\mathbf{X}^{<t-\tau_i,t>}$) and points from the reference path ($\mathbf{Z}_{ref}^{<t-\tau_i,t+\tau_o>}$). Future trajectory points ($\mathbf{Y}^{<t+1,t+\tau_o>}$) are used to calculate the labels in a self-supervised manner. The encoder-decoder network architecture is designed to take advantage of the neural network's ability to learn effective temporal representations. The input sequences are passed through an encoder network, which maps raw inputs to a hidden feature representation known as a *thought vector*, and a decoder network, which takes this feature representation as input, processes it, and outputs a trajectory prediction.

As opposed to Reference [4], we now sense the world in 3D using an octree representation, and we no longer use convolutional layers for processing the input sequences, as this intermediate representation has been taken over by the fixed state vector between the encoder and the decoder of our architecture.

The key contributions of this paper are:

1. based on the octree environment model, we provide a solution for estimating local driving trajectories by reformulating the estimation task as a classification problem with a configurable resolution;
2. we define an encoder-decoder deep neural network topology for predicting desired future trajectories, which are obtained in a self-supervised fashion;
3. we leverage the innate property of the state vector between the encoder and the decoder to represent a learned sequence of trajectory points constrained by road topology.

The rest of this paper is organized as follows. Section 2 covers the related work. In Section 3, the trajectory estimation problem for mobile robots is briefly described (Section 3.1), together with the octree environment model (Section 3.2) and the Rovis-

Lab's AMTU kinematic model (Section 3.3). Our choice of recurrent encoder-decoder neural network architecture (Section 4.1) is detailed in Section 4, in combination with the training setup (Section 4.2) using the gathered mobile robot data. Performance evaluation can be found in Section 5, where we first overview the experimental setup (Section 5.1) and detail the related experiments (Sections 5.2 and 5.3), and second, we analyze the deployment of our OctoPath network on the NVidia AGX Xavier (Section 5.5) and an ablation study of varying the octree resolution and the network model parameters (Section 5.4). Sections 6 covers the conclusions of the paper.

2. Related Work

Deep learning [5,6] has emerged as the driving force behind many of the new developments in recent years, with notable advancements in computer vision, robotics, and natural language processing. DNNs have effectively learned representations that generalize well for different scenarios arising from real data when applied to a variety of machine learning tasks.

Among the different kinds of DNN architectures available, recurrent neural networks (RNN) are used to analyze the structure of time series data, such as text or video streams [7]. All of these advancements in the field of deep learning have obviously influenced the growth of intelligent vehicles. The first automated cars were described in the late 1980s and early 1990s, and following these first implementations, various control architectures for automated driving were proposed [8]. The driving functions are generally applied as perception-planning-action pipelines, as seen in Grigorescu et al.'s deep learning for autonomous driving survey [9], but recent approaches based on end2end learning have also been proposed, though most as research prototypes.

In this context, end2end learning is defined as developing and training a complex neural network to directly map input sensory data to vehicle commands [10]. The authors of Reference [11] present an end-to-end imitation learning system for off-road autonomous driving by using only low-cost onboard sensors, having their DNN policy trained for agile driving on a predefined obstacle-free track. Since self-driving cars must manage roads with complex barriers and unclear lane borders, this strategy restricts the applicability of their system to autonomous driving. In the PilotNet algorithm proposed by Bojarski et al. [12], the input images are directly mapped to the vehicle's steering control.

Because large annotated datasets are needed to train such deep networks, the alternative of self-supervised learning approaches has also been employed. In Reference [13], BADGR—Berkeley Autonomous Driving Ground Robot, an End2End self-supervised learning system, was created to navigate in real-world situations with geometrically distracting obstacles (such as tall grass). It can also take into account terrain preferences, generalize to new environments, and improve on its own by collecting more data.

DRL (Deep Reinforcement Learning) is an algorithm or a system in which agents learn to act by communicating with their surroundings. Although DRL does not use training data, it maximizes a cumulative reward that quantifies its behavior [14]. In Reference [15], well-established route planning techniques are correlated with novel neural learning approaches to find the best path to a target location within a square grid. A combination of global guidance and a local RL-based planner is presented in Reference [16]. A major downside of DRL is that such an architecture is difficult to train in real-life scenarios due to the interaction constraint and tends to generalize on specific driving scenarios (e.g., highway driving). Furthermore, because the input from the sensors is straightly mapped to actuators, these systems' functional safety is normally difficult to monitor [17].

Path planning in mobile robotics has been a topic of research for decades [18], and it is categorized into global and local planning based on the scope and executability of the plan [19]. The survey in Reference [3] offers a broad overview of route planning in the automotive industry. It focuses on the task planner, behavior planner, and motion planner, which are all taxonomy elements of path planning. It does not, however, provide a study

of deep learning technology, despite increased interest in using deep learning technologies for route planning and behavior arbitration in the state-of-the-art literature.

Novel DNN-based path planning methods which have been developed are based on biologically inspired cognitive architectures [20], with the three primary methods being swarm intelligence, evolutionary algorithms, and neurodynamics. In the case of Reference [21], an optimal path planning algorithm based on convolutional neural networks (CNN) and random-exporing trees (RRT) is presented. Their approach, called Neural RRT*, is a framework for generating the sampling distribution of the optimal path under several constraints. In our previous work on local state trajectory estimation [4], we used a multi-objective neuro-evolutionary approach to train a regression-based hybrid CNN-LSTM architecture using sequences of 2D occupancy grids.

Since motion planning can also be viewed as a sequence to sequence mapping problem, or as a sequence generation task, RNNs have been proposed for modeling the driving trajectories [22–24]. The input sequence in this technique is composed of the most recent sensor readings, while the output sequence contains the future trajectory points. In contrast to traditional neural networks, RNN memory cells have a time-dependent feedback loop. In order to use RNNs for predicting a future trajectory, each separate point is considered a state, which further implies that the whole trajectory is represented as a sequence. The transition from one state to another is strictly constrained by the topology of the network [25].

Many approaches address the task of predicting the trajectory of vehicles surrounding the ego-car: in Reference [26], the authors suggested a multi-modal trajectory prediction system for surrounding vehicles, which assigns confidence values to vehicle maneuvers and generates a multi-modal distribution over future motion based on those values. The authors of Reference [27] suggest an LSTM that predicts the location of cars in an occupancy grid at discrete intervals of 0.5 s, 1 s, and 2 s in the future, while Park et al. [23] employs an encoder-decoder architecture to generate the *K* most likely trajectory candidates over an occupancy grid map using the beam search method.

Most of the RNN solutions proposed for solving the task of trajectory estimation need a discrete environment model. In this work, the proposed environment model is based on octrees [28,29] and uses probabilistic occupancy estimation. The main advantages of using this model are that it explicitly represents not only occupied space but also free and unknown areas and that it enables a compact memory representation and configurable resolutions. In Reference [30], the authors used an octree-based model to determine the surrounding obstacle locations in real-time, and use it for path planning and robot motion generation. The authors of Reference [31] present the advantages of having an environment with multiple resolutions and a uniform octree representation mechanism of models from various sensors.

3. Method

A variable's value is defined either as a discrete sequence defined in the $< t, t + k >$ time interval, where k represents the length of the sequence, or as a single discrete time step t, written as superscript $< t >$. The value of a trajectory output variable $\mathbf{y}$, for example, may be specified at discrete time t as $\mathbf{y}^{<t>}$ or within a sequence interval $\mathbf{Y}^{<t,t+k>}$.

3.1. Problem Definition: Local Trajectory Prediction

A trajectory is defined as a time-and-velocity-parameterized sequence of states visited by the vehicle. Local trajectory planning (also known as local trajectory generation) is concerned with the real-time planning of a vehicle's transition from its current state to the next while avoiding obstacles and meeting the vehicle's kinematic limitations, over a prediction horizon. Depending on the speed and line-of-sight of the vehicle's on-board sensors, the route planner module produces an estimated optimal trajectory from the vehicle's current location, with a look-ahead distance, during each planning cycle.

Figure 2 depicts an illustration of the local state trajectory prediction task for autonomous driving. The task is to learn a local trajectory for navigating the ego-vehicle to

destination coordinates $\mathbf{y}_{dest}^{<t+\tau_o>}$ given a sequence of octrees $\mathbf{x}^{<t-\tau_i>} : \mathbb{R}^3 \times \tau_i$, the current sequence from the reference route $\mathbf{z}_{ref}^{<t-\tau_i,t+\tau_o>} : \mathbb{R}^2 \times \tau_i$, the position of the ego-vehicle $\mathbf{p}_{ego}^{<t>} \in \mathbb{R}^2$ in $\mathbf{x}^{<t>}$, and the destination coordinates $\mathbf{p}_{dest}^{<t>} \in \mathbb{R}^2$ at time t. The length of the input data sequence is τ_i, and τ_o is the number of time steps for which the ego-vehicle's trajectory is computed. The reference path is represented by a collection of points in the global coordinate system, which depict the route that the robot has to follow.
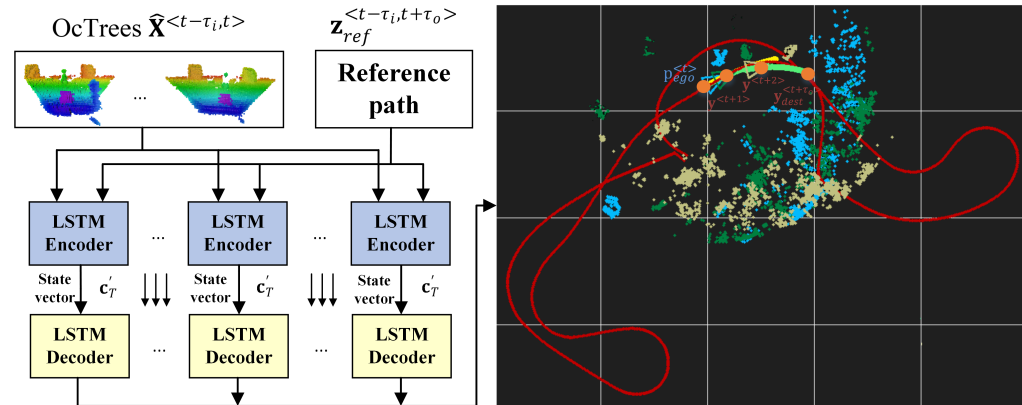

Figure 2. Estimating local trajectories for autonomous robots and vehicles using encoder-decoder recurrent neural networks. Considering the current ego vehicle's position $\mathbf{p}_{ego}^{<t>}$, an input sequence of octrees $\mathbf{X}^{<t-\tau_i,t>} = [\mathbf{x}^{<t-\tau_i>}, \ldots, \mathbf{x}^{<t>}]$, the current sequence from the reference route $\mathbf{z}_{ref}^{<t-\tau_i,t+\tau_o>}$ and a desired destination from the reference path $\mathbf{y}_{dest}^{<t+\tau_o>}$, the aim is to estimate a driving trajectory $\mathbf{Y}^{<t+1,t+\tau_o>} = [\mathbf{y}^{<t+1>}, \ldots, \mathbf{y}_{dest}^{<t+\tau_o>}]$.

The local trajectory problem can be expressed as a classification problem with N classes, where N is determined by the OcTree resolution (density of points in the environment space) and the prediction horizon (how far away is the destination point). It is a multi-class classification problem in which each time step's point on the trajectory is selected sequentially from an input sequence of octree environment snapshots and points from the reference route.

In other words, we pursue a desired local navigation trajectory of the ego-vehicle from any arbitrary starting point $\mathbf{p}_0^{<t>}$ (which is a coordinate in the current input octree) to $\mathbf{y}_{dest}^{<t+\tau_o>}$, with the following properties:

- the longitudinal velocity $v^{<t,t+\tau_o>}$ is maximal and is contained within the bounds $[v_{min}, v_{max}]$;
- the total distance between consecutive trajectory points is minimal: $||(p_0^{<t>} - \mathbf{y}_{dest}^{<t>}) + \sum_{i=0}^{\tau_0}(\mathbf{y}_{dest}^{<t+i>} - \mathbf{y}_{dest}^{<t+i+1>})||$;
- the lateral velocity $v_\delta^{<t,t+\tau_o>}$ is minimal. It is is determined by the rate of change of the steering angle $v_\delta \in [\dot{\delta}_{min}, \dot{\delta}_{max}]$.

The vehicle is modeled based on the kinematics of a skid-steering wheeled mobile robot, with position state $\mathbf{p}^{<t>} = (p_x^{<t>}, p_y^{<t>})$ and no-slip assumptions, which is further detailed in Section 3.3.

3.2. Octree Environment Model

Most robotic applications require an environment model, which must be effective in terms of runtime and memory usage, and include free, occupied, and unmapped zones. Sensor models often have range measurement errors, and there may also be apparently random measurements caused by reflections or dynamic barriers. The underlying uncertainty must be taken into account when creating an accurate model of the environment from such noisy data. Multiple uncertain measurements can then be combined to form a reliable estimation of the environment's true state.

An octree is a hierarchical data structure for 3D spatial subdivision that is most commonly used to partition a given 3D space into eight octants by recursively subdividing it. Every node on an octree is the space of a cubic volume that is commonly referred to as a voxel. Each internal node has exactly eight children, and the octree's *resolution* is determined by the minimum voxel scale. If the inner nodes are retained accordingly, the tree can be cut down at any level to acquire a more coarse subdivision. Octrees prevent one of the key weaknesses in fixed grid systems in robotic mapping, the fact that the surroundings should not previously be known, and that the environmental model includes only the assessed volume.

Octrees prevent one of the key weaknesses in fixed grid systems in robotic mapping, the fact that the mapping surroundings should not previously be known, and the environmental model includes only volume measured.

When referring to a laser range finder, for example, the endpoints of the sensor generate occupied space, while the detected region between the sensor and the endpoint is considered to be free space. The occupied space is mapped from the point cloud data packets at the corresponding distance in space for our input LiDAR data. As a result, we use LiDAR data to generate an octree environment model, which depicts free-space (driving area) and inhabited areas in three dimensions.

A central property of our approach is that it allows for efficiency of occupied and free space while keeping the memory consumption low, which is essential for our model car hardware from Figure 3. The octrees have fixed sizes, as required by the neural network input, based on the field of view of the LiDAR sensor. The nodes which are neither occupied nor free (these are always beyond the detected obstacles) are marked as unknown and initialized with zero 0 to prevent them from influencing the inference result. Additionally, we can configure the resolution to a lower value, to reduce the processing times and memory usage even further. Details regarding the impact of varying the octree resolution are provided in Section 5.4
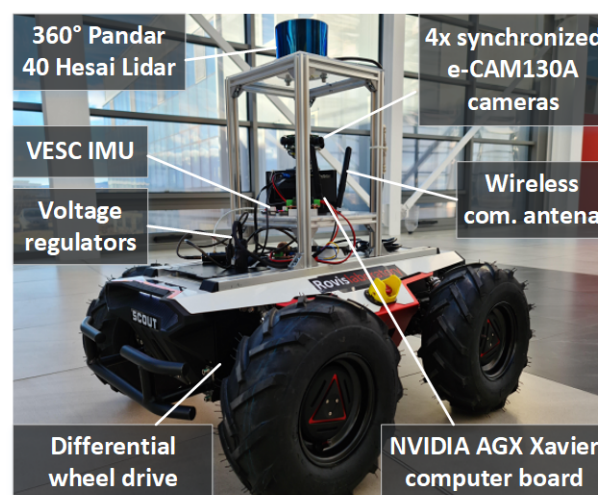

Figure 3. RovisLab AMTU (Autonomous Mobile Test Unit). The robot is a SSWMR (skid-steer wheeled mobile robot) platform equipped with a 360 degree, 40-Channel Hesai Pandar Lidar, 4× e-CAM130A cameras, a Tinkerforge inertial measurement unit (IMU), and an NVIDIA AGX Xavier computer board for real-time data processing and control.

3.3. Kinematics of RovisLab's AMTU as a SSWMR (Skid-Steer Wheeled Mobile Robot)

Figure 4 shows the schematic diagram of a SSWMR (skid-steer wheeled mobile robot) [32], together with a top view of RovisLab's AMTU. The following model assumptions are taken into account:

1. The robot's mass center is at the geometric center of the body frame;
2. Each side's two wheels rotate at the same speed;

3. The robot is operating on a firm ground floor with all four wheels in contact with it at all times.

As shown in Figure 4, we define an inertial frame (X, Y) (global frame) and a local (robot body) frame (x, y). Presume the robot movies in a plane with linear velocity $v = (v_x, v_y, 0)^T$ and rotates with an angular velocity $\omega = (0, 0, \omega_z)^T$, both expressed in the local frame. If $q = (X, Y, \theta)^T$ is the state vector defining the robot's generalized coordinates (position X and Y, as well as the orientation θ of the local coordinate frame with respect to the inertial frame), then $\dot{q} = (\dot{X}, \dot{Y}, \dot{\theta})^T$ is the vector of generalized velocities.

The relationship between the robot velocities in both frames is then calculated as follows:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix}. \tag{1}$$
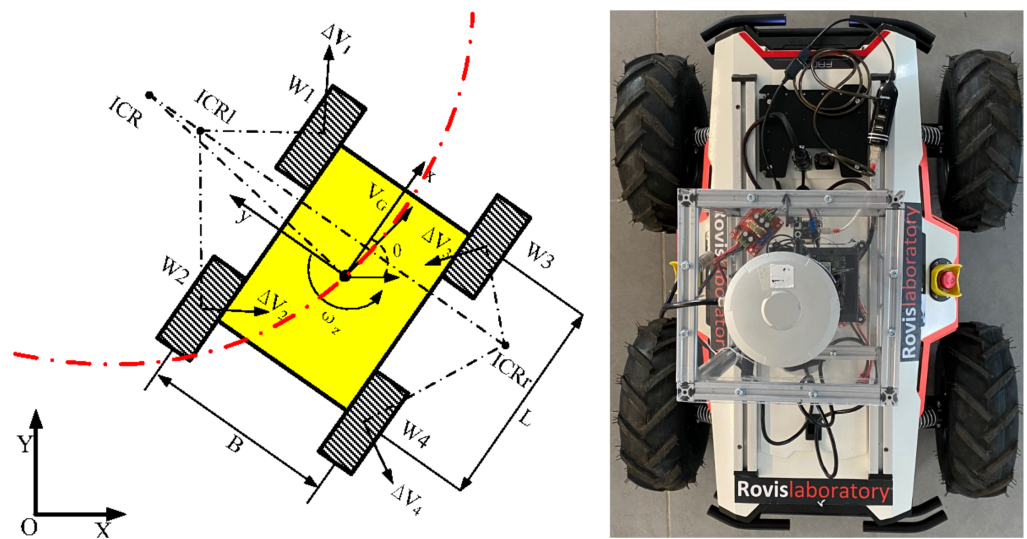

Figure 4. A skid-steering mobile robot's kinematics diagram and the top view of RovisLab Autonomous Mobile Test Unit.

Because it only specifies free-body kinematics, Equation (1) places no limits on the SSWMR plane movement. As a result, the relationship between wheel velocities and local velocities must be analyzed. For simplicity, the thickness of the wheel is neglected and is assumed to be in contact with the plane at point P_i, as according to the initial model assumption nr. 3. In comparison to other wheeled vehicles, the SSWMR has a non-zero lateral velocity. This property stems from the SSWMR's mechanical structure, which necessitates lateral skidding if the vehicle's orientation shifts. As a result, the wheels are only tangent to the path when $\omega = 0$, that is, when the robot travels in a straight line. It is important to consider all wheels together when developing the kinematic model.

Let ω_i and v_i, with $i = 1, 2, 3, 4$ denote the wheel angular and center linear velocities for front-left, rear-left, front-right, and rear-right wheels, respectively. Thus, we have:

$$\omega_L = \omega_1 = \omega_2, \qquad \omega_R = \omega_3 = \omega_4. \tag{2}$$

We can use the previous equation to state the direct kinematics on the plane:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = f \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix}, \tag{3}$$

where $v = (v_x, v_y)$ is the vehicle's translational velocity with respect to its local frame, ω_z is its angular velocity, and r is the radius of the wheel.

The instantaneous centers of rotation (ICR) of the left-side, right-side, and robot body are denoted as ICR_l, ICR_r, and ICR_G, respectively, while the mobile robot moves. ICR_l, ICR_r, and ICR_G are all known to lie on a line parallel to the x-axis [33]. We define the x-y coordinates for ICR_l, ICR_r, and ICR_G as (x_{ICR_l}, y_{ICR_l}), (x_{ICR_r}, y_{ICR_r}), and (x_{ICR}, y_{ICR}), respectively. The sides' angular velocity is equal to the velocity of the robot body ω_z. We further obtain the following geometrical relations:

$$x_{ICR} = x_{ICR_l} = x_{ICR_r} = -\frac{v_y}{\omega_z} \tag{4}$$

$$y_{ICR} = \frac{v_x}{\omega_z}, \tag{5}$$

$$y_{ICR_l} = \frac{v_x - \omega_l r}{\omega_z}, \tag{6}$$

$$y_{ICR_r} = \frac{v_x - \omega_r r}{\omega_z}. \tag{7}$$

From Equations (4)–(7), the kinematics relation (3) can be represented as:

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = J_\omega \begin{bmatrix} \omega_l r \\ \omega_r r \end{bmatrix}, \tag{8}$$

where the elements of matrix J_ω are determined by the ICR coordinates on the left and right sides:

$$J_\omega = \frac{1}{y_{ICR_l} - y_{ICR_r}} \begin{bmatrix} -y_{ICR_r} & y_{ICR_l} \\ x_{ICR} & -x_{ICR} \\ -1 & 1 \end{bmatrix}. \tag{9}$$

Since the SSWMR is symmetrical in our case, we can obtain a symmetrical kinematics model, as seen on the right side of Figure 4. As a result, the ICRs are symmetrically distributed on the x-axis, and the matrix J_ω can be written as follows:

$$J_\omega = \frac{1}{2y_{ICR_0}} \begin{bmatrix} y_{ICR_0} & y_{ICR_0} \\ 0 & 0 \\ -1 & 1 \end{bmatrix}, \tag{10}$$

where $y_{ICR_0} = y_{ICR_l} = -y_{ICR_r}$ represents the side ICR values. Considering that, for our symmetrical model, $v_l = \omega_l r$ and $v_r = \omega_r r$, the relations between the angular wheel velocities and the robot velocities are as follows:

$$\begin{cases} v_x = \frac{\omega_l r + \omega_r r}{2} = \frac{v_l + v_r}{2} \\ v_y = 0 \\ \omega_z = \frac{-\omega_l r + \omega_r r}{2y_{ICR_0}} = \frac{-v_l + v_r}{2y_{ICR_0}} \end{cases}. \tag{11}$$

Based on Equation (11), the control signal u can be written as:

$$u = \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} = r \begin{bmatrix} \frac{\omega_l + \omega_r}{2} \\ \frac{-\omega_l + \omega_r}{2y_{ICR_0}} \end{bmatrix}. \tag{12}$$

The last equation shows that the pair of angular velocities ω_l and ω_r, as well as velocities v_x and ω_z, can technically be viewed as a control kinematic input signal. The accuracy of relation (12), on the other hand, is heavily reliant on longitudinal slip, and it can only be used if this phenomenon is not dominant. Furthermore, the parameters r and y_{ICR_0} can be calculated experimentally to ensure that the angular robot velocity is accurately estimated in relation to the angular velocities of the wheels.

4. OctoPath: Architecture, Training and Deployment

4.1. RNN Encoder-Decoder Architecture

In contrast to traditional neural networks, an RNN's memory cell comprises a time-dependent feedback loop. A recurrent neural network itself can be "unrolled" $\tau_i + \tau_o$ times to produce a loop-free architecture that matches the input length, if we consider an input sequence $[x^{<t-\tau_i>}, \ldots, x^{<t>}]$ which is time dependant, together with an output sequence $[y^{<t+1>}, \ldots, y^{<t+\tau_o>}]$. Unrolled networks have $\tau_i + \tau_o + 1$ similar or even identical layers, which means that each layer has the same learned weights.

This architecture is comprised of two models: a stack of several recurrent units for reading the input sequence and encoding it into a fixed-length vector, and a second one for decoding the fixed-length vector and outputting the predicted sequence. The combined models are known as an *RNN Encoder-Decoder*, which is designed specifically for sequence to sequence problems. Given the input sequence $\mathbf{X}^{<t-\tau_i, t>}$, a basic RNN encoder computes the sequence of hidden states (h_1, h_2, h_3, ..., h_N):

$$h_t = \tanh(U_{xh}x_t + U_{hh}h_{t-1}), \tag{13}$$

where the two matrices U_{xh} and U_{hh} are the weight matrix between the input layer and hidden layer, and the weight matrix of recurrent connections in a given hidden layer, respectively.

The vanishing gradient experienced during training is the major challenge when using simple RNNs. The gradient signal can be multiplied an infinite number of times, up to the number of time steps. As a result, a classical RNN cannot capture long-term dependencies in sequence data. The gradient of the network's output will have a hard time propagating back to affect the weights of the earlier layers if the network is very deep or processes long sequences. The weights of the network will not be successfully modified as a result of gradient vanishing, resulting in very small weight values.

To counter these challenges, in our work, we use a set of Long Short-Term Memory (LSTM) networks for both the encoder and the decoder, as shown in Figure 1. LSTMs solve the vanishing gradient problem by adding three gates that control the input, output, and memory state, as opposed to classical recurrent neural networks.

$\Theta = [W_i, U_i, b_i]$ parametrizes an LSTM network, where W_i embodies the weights of the gates and memory cells multiplied with the input state, U_i represents the weights controlling the network's activations, and b_i contains the bias values of the neurons. A network output sequence is defined as a desired ego-vehicle optimal trajectory:

$$Y^{<t+1, t+\tau_o>} = [y^{<t+1>}, y^{<t+2>}, \ldots, y^{<t+\tau_o>}], \tag{14}$$

where $y^{<t+1>}$ is a predicted trajectory set-point at time $t + 1$. τ_i and τ_o are not necessarily equal: $\tau_i \neq \tau_o$.

The LSTM encoder takes the latest octree samples $\mathbf{X}^{<t-\tau_i, t>}$, as well as the reference trajectory sequence $\mathbf{Z}_{ref}^{<t-\tau_i, t+\tau_o>}$ for the current time step t, and produces an intermediate fixed-size vector c_t that preserves the temporal correlation of the previous observations. The hidden state of the LSTM encoder h_t is calculated using the following equations:

$$z_t = \sigma(U_{xz}x_t + U_{hz}h_{t-1}), \tag{15}$$

$$r_t = \sigma(U_{xr}x_t + U_{hr}h_{t-1}), \tag{16}$$

$$\tilde{h}_t = \tanh(U_{xh}x_t + U_{rh}(r_t \otimes h_{t-1})), \tag{17}$$

$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h}_t, \tag{18}$$

where σ represents the sigmoid activation function. z_t, r_t, and $\tilde{h}_t$ are the update gate, reset gate, and candidate activation, respectively. U_{xz}, U_{xr}, U_{xh}, U_{hz}, U_{hr}, and U_{rh} are the related weight matrices. The notation $\otimes$ represents an element-wise multiplication operator.

The LSTM decoder takes the predicted trajectory sample to produce the subsequent trajectory samples, producing the entire future trajectory $\mathbf{Y}^{<t+1,t+\tau_o>}$ for the current time step, given the context vector c_t as input. $\mathbf{Y}^{<t+1,t+\tau_o>}$ is defined as a sequence variable Y with data instances $[y^{<t+1>}, \ldots, y^{<t+\tau_o-1>}, x^{<t+\tau_o>}]$ in a specific time interval $[t+1, t+\tau_o]$. Each predicted sequence variable's probability is calculated as follows:

$$p(y_t | X, y_{t-1}) = g(U_o(Ey_{t-1} + U_s s_t + U_c c_t)), \tag{19}$$

where g is a softmax activation function, as it can be observed in Figure 1. s_t is the current hidden state of the decoder, and y_{t-1} represents the previous target symbol, while E denotes the embedding matrix.

The earlier target sequence variable y_{t-1} and the context vector ct are also inputs to the decoder, which uses a single unidirectional layer to compute the hidden state st:

$$z'_t = \sigma(U_{yz} E y_{t-1} + U_{sz} s_{t-1} + C_{cz} c_t), \tag{20}$$

$$r'_t = \sigma(U_{yr} E y_{t-1} + U_{sr} s_{t-1} + C_{cr} c_t), \tag{21}$$

$$\tilde{s}_t = \tanh(U_{ys} E y_{t-1} + U_{rs}(r'_t \otimes s_{t-1}) + C_{cs} c_t), \tag{22}$$

$$s_t = (1 - z'_t) \otimes s_{t-1} + z'_t \otimes \tilde{s}_t, \tag{23}$$

where z'_t, r'_t, and $\tilde{s}_t$ are the update gate, reset gate, and candidate activation, respectively. U_{xx} and C_{xx} are the related weight matrices.

The decoder retains the best sequence candidates in the algorithm when creating the future trajectory sample for each time step. As a result, using the octree input framework, the proposed model would predict the most likely hypotheses of the vehicle trajectory. As analogy to machine translation problems, a point coordinate inside an octree is a character, an octree is a word, and the sequence of input octrees represents an sentence. Our experiments show that an encoder-decoder RNN produces an acceptable trajectory and that its prediction accuracy is improved in comparison to traditional prediction methods.

4.2. Training Setup

To train the network, we used data collected with RovisLab AMTU, with the robot being manually driven in the test environment while encountering various static and dynamic obstacles. When acquiring training samples, the point cloud sensory data acquired over Ethernet as UDP packets from a Hesai Pandar LiDAR is used to populate octrees $\mathbf{X}^{<t-\tau_i,t>}$. Afterwards, together with the global reference trajectory $\mathbf{z}_{ref}^{<t-\tau_i,t+\tau_o>}$ and with the future position states $\mathbf{Y}^{<t-\tau_i,t>}$, these are stored as sequence data. For practical reasons, the global reference trajectory is stored at sampling time t over the finite horizon $[t - \tau_i, t + \tau_o]$.

A single training sample, combined from the snapshot of an octree generated from input LiDAR data, maps to $\mathbf{X}^{<t>}$ and a continuous sequence of octrees is represented further as $\mathbf{X}^{<t-\tau_i,t>}$. Figure 5 shows an example of input LiDAR data and afferent octree representations queried for occupied voxels, together with the projected 2D environment model. A script implementation is also necessary for processing the sequences from the acquired input data. There is a large number of configurable parameters, such as the sampling time, resolution of the camera, rotations per minute of the LiDAR, and coordinate system for the ego-vehicle data.

We train our method using a self-supervised learning approach; therefore, we do not require any manual labeling of the training data. In order to mitigate overfitting, the obstacles were placed differently each time. During training, we have used an 80/10/10 train-validation-test-data split. OctoPath has been trained for 15,000 epochs in a self-supervised fashion using the stochastic gradient decent method with the Adam optimizer [34], with a learning rate of 0.0003. As presented in Section 3.1, we express the local trajectory estimation task as a multi-classification problem with N classes (given by the OcTree resolution); thus, the loss function to be minimized is given by the negative log-likelihood (NLL) function, as we also have a softmax layer in the last layer of our network. The learning

curve of our network (between the training and validation set) can be found on the right side of Figure 8.

We used the following hardware configuration to decrease the time required for training: two high-performance graphics cards, namely Nvidia GeForce RTX2080Ti, connected with NVLink, managed by an Intel Core i9-9900K CPU, 64 GB RAM, and a 1TB SSD. For the training itself, we have used TensorFlow, because it has seamless integration with the Keras API and is maintained in Python.

5. Results

5.1. Experimental Setup Overview

OctoPath was compared to the baseline hybrid A* algorithm [35], to a regression-based approach [25], and to a CNN learning-based approach [21]. We put the OctoPath algorithm to the test in two distinct environments: (I) in the GridSim simulator [36] (More information is available at www.rovislab.com/gridsim.html, accessed on 21 May 2021.) (Section 5.2) and (II) in a real-world navigation environment, both indoor and outdoor (Section 5.3), using the RovisLab AMTU robot from Figure 3. RovisLab AMTU is an AgileX Scout 2.0 platform which acts as a 1:4 scaled car, equipped with a 360° Hesai Pandar 40 Lidar, 4× e-130A cameras providing a 360° visual perception of the surroundings, a VESC inertial measurement unit, GPS, and an NVIDIA AGX Xavier board for data processing and control. The state of the vehicle was measured using wheels odometry and the Inertial Measurement Unit (IMU).

All experiments aimed at solving the trajectory estimation problem illustrated in Figure 2, which was to calculate a trajectory for safely navigating the driving environment without performing the motion control task. To implement motion control, the predicted states were used as input to a model predictive controller, which computed the necessary v_x and ω_z control signals for the RovisLab AMTU, as detailed in Section 3.3. The motion controller's design and implementation are beyond the scope of this paper.

The hybrid A* algorithm employs a modified state-update rule to apply a variant of the well-known A* algorithm to the vehicle's octree environment model. The search space (x, y, θ) is discretized, just like in traditional A*, but unlike A*, which only allows visiting cell centers, the hybrid version of the algorithm associates a more continuous state of the car with each grid cell, allowing also trajectory points that are not in the exact center of the octree cell.

In the case of trajectory prediction as a regression problem, the goal is to achieve a direct prediction of continuous future positions without any discretization. Because the average prediction minimizes the regression error, such methods have a bias to output the average of several options, thus rendering it inaccurate.

The Neural RRT* algorithm, proposed by Wang et al. in Reference [21], is a novel optimal path planning algorithm based on convolutional neural networks. It used the A* algorithm to generate training data, considering map information as input, and the optimal path as ground truth. Given a new path planning problem, the model can quickly determine the optimal path's probability distribution, which is then used to direct the RRT* planner's sampling operation. The performance of the algorithm varies under different values of the clearance to the obstacles and step size. A wider clearance indicates that the planned route is far from the obstacles, while a smaller clearance indicates that the planned path is closer to them. We have used a fixed step size of 2 and a robot clearance value of 4.

We use the Root Mean Square Error (RMSE) between the predicted and the recorded trajectory in the 2D driving plane:

$$RMSE = \sqrt{\frac{1}{\tau_o} \sum_{t=1}^{\tau_o} \left[(\hat{p}_x^{<t>} - p_x^{<t>})^2 + (\hat{p}_y^{<t>} - p_y^{<t>})^2 \right]}, \tag{24}$$

where $\hat{p}_x^{<t>}$, $\hat{p}_y^{<t>}$ are the points on the predicted trajectory, and $p_x^{<t>}$, $p_y^{<t>}$ are the points on the ground truth trajectory, respectively. We set the prediction horizon $\tau_o = 10$.

The workflow of the experiments is as follows:

- collect training data from driving recordings;
- generate octrees and format training data as sequences;
- train the OctoPath deep network from Figure 1;
- evaluate on simulated and real-world driving scenarios.

This experimental setup resulted in 15 km of driving in GridSim, over 1 km of looped indoor navigation and over 2 km of outdoor navigation outside of Transilvania University of Brasov's IHTPSD (Institute of High Tech Products for Sustainable Development). The robot navigated indoor and outdoor environments while avoiding static and dynamic obstacles.

5.2. Experiment I: GridSim Simulation Environment

GridSim is a self-driving simulation engine that generates synthetic occupancy grids from simulated sensors using kinematic models, which are then used to produce input octree data. The user interface was integrated into the GridSim environment menu, such that the modes can be switched between replay, record, and training, with each one having access to the five different scenarios. There is a large number of configurable parameters, such as the resolution of the simulator, occupancy grid precision, number of traffic participants, ego vehicle's size, maximum speed, or turning radius.

The first set of experiments compared the four algorithms discussed in Section 5.1 over 15 km of driving in GridSim [36]. The goal is to get from a starting position to a given destination while avoiding collisions and driving at the desired speed. The Z coordinate of all obstacle and free space points is set to zero to adjust the encoder-decoder network's input data to the GridSim environment. The testing scenarios generated using the GridSim simulation environment were not used during the training of the network.

For the various types of roads and traffic environments found in the synthetic testing database, the performance assessment of the benchmarked algorithms is summarized in the top part of the Table 1. We illustrate the mean position errors ($\bar{e}_x$, $\bar{e}_y$), as well as the RMSE metric from Equation (24).

Table 1. Errors between estimated and ground truth trajectories in simulation and real-world navigation testing scenarios.

Scenario	Method	$\bar{e}_x[m]$	$\max(e_x)[m]$	$\bar{e}_y[m]$	$\max(e_y)[m]$	$RMSE[m]$
GridSim simulation	Hybrid A*	1.43	3.21	2.71	4.01	2.71
	Regression	3.51	7.20	4.71	8.53	5.10
	Neural RRT	1.27	3.01	2.35	2.98	2.48
	Octopath	1.16	2.31	1.72	2.75	2.07
Indoor navigation	Hybrid A*	1.21	4.33	1.33	3.88	1.74
	Regression	1.90	5.73	2.31	4.98	2.75
	Neural RRT	1.01	3.29	0.98	2.16	1.44
	Octopath	0.55	1.08	0.44	0.87	0.69
Outdoor navigation	Hybrid A*	1.35	4.67	1.44	4.44	1.98
	Regression	2.41	8.42	2.77	8.98	3.01
	Neural RRT	1.05	2.52	1.06	3.24	1.17
	Octopath	0.71	1.46	0.57	1.17	0.88

5.3. Experiment II: Indoor and Outdoor Navigation

The indoor navigation experiment was performed using the RovisLab AMTU SSWMR vehicle from Figure 3, with different indoor navigation tasks. The reference routes which the car had to follow were composed of straight lines, S-curves, circles, and a 75 m track on the main hallway of Transilvania University of Brasov's Institute for Research, as it can be observed in Figure 5.
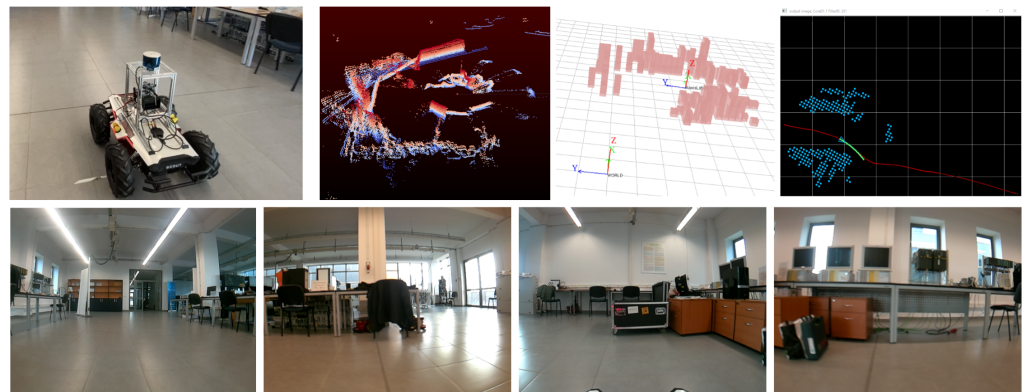
Figure 5. Indoor testing setup. (top) RovisLab AMTU on the reference trajectory, with raw LiDAR data acquired from the top-mounted sensor and the generated 3D and projected 2D octree environment model. (bottom) The 4 images were acquired with the e-CAM130A quad camera. The order of the images is, given the camera mounting position, front-right-back-left.

The testing room for the indoor experiment was the same as the one that was used for gathering training data, but the reference routes and the obstacles were placed differently. The main hallway was not used for gathering training data.

The first set of 10 trials were performed without any obstacles present on the reference routes, while the second 10 trials set contained static and dynamic obstacles. Fifty-four thousand training samples have been collected in the form of LiDAR data and vehicle states, as seen in Figure 5. The path driven when collecting data was considered as a reference trajectory and was created in a self-supervised manner.

The outdoor navigation experiment was performed outside of Transilvania University of Brasov's IHTPSD (Institute of High Tech Products for Sustainable Development), as it can be observed in Figure 6. The reference route which the car had to follow was composed of a full loop around the institute and was created using a GPS tool. The route itself is around 500 m long, and we ran it 4 times.
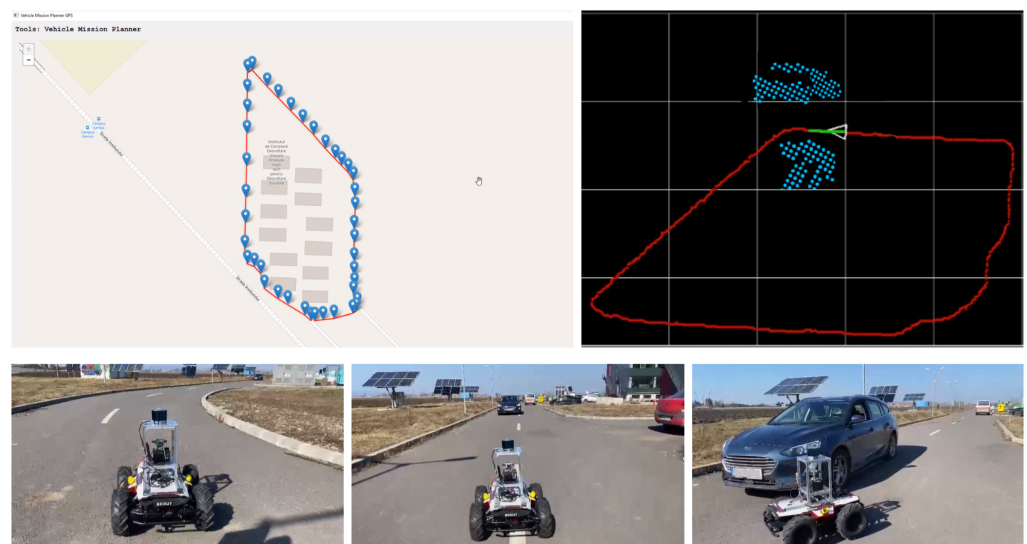

Figure 6. Outdoor testing setup. (**top-left**) Our vehicle mission planner tool generating the GPS reference route. (**top-right**) RovisLab AMTU on the reference trajectory with the projected 2D octree environment model. (**bottom**) On-route behavior with dynamic obstacle avoidance.

The outdoor reference path which was used for training the network was recorded as the driven path when collecting the sensory data. When testing the network, the reference path was generated using our vehicle mission planner tool which can be seen in Figure 6.

The static obstacles were mainly parked cars, while the dynamic obstacles were moving cars or people.

The mean and standard deviation of the position error (computed as RMSE) can be viewed in Figure 7, left side for indoor navigation, and right side for outdoor navigation. The position errors are shown in Table 1 for all scenarios: simulation, indoor navigation and outdoor navigation. The mean ($\bar{e}_x$, $\bar{e}_y$) and maximum ($\max(e_x)$, $\max(e_y)$) position errors, as well as the RMSE metric from Equation (24), are shown. When compared to OctoPath, Neural RRT has the lowest deviations, but, from the non-learning approaches, Hybrid A* performs the best, indicating that it is a good candidate for non-learning trajectory estimation.
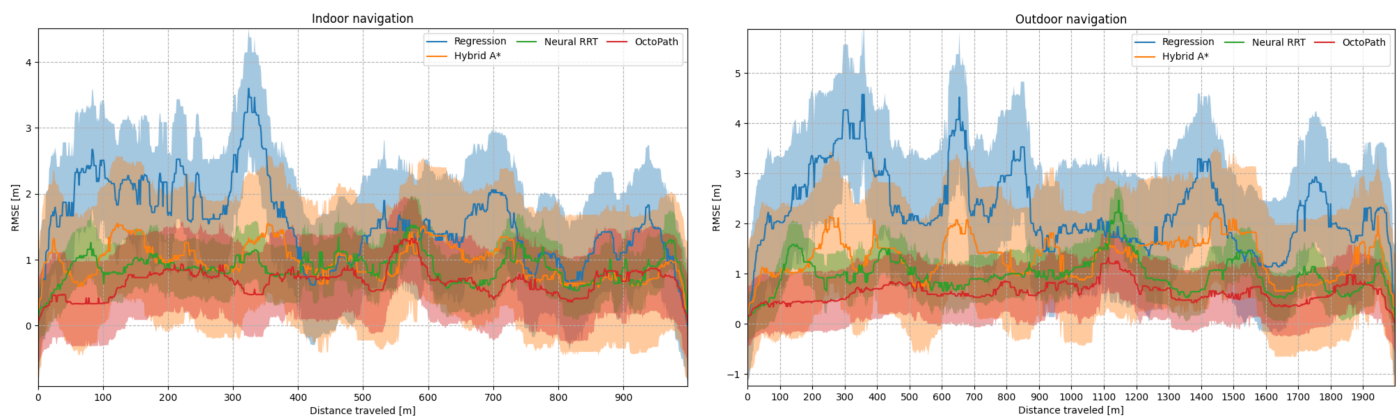

Figure 7. Mean (solid line) and standard deviation (shaded region) of the position error, computed as the RMSE from Equation (24) during indoor and outdoor testing scenarios.

5.4. Ablation Study

In this section, we analyze the impact of varying the octree resolution and the network model parameters, as well as the length of the input data sequence from the annotated trajectory dataset. As in the previous section, the RMSE between the estimated and the ground truth driven trajectory is used to quantify the accuracy of our system with respect to the variation of these parameters.

In the first development stage of our algorithm, we have varied the number of LSTM layers inside both the encoder and the decoder, while keeping a fixed number for the output sequence. The experiments testing errors were averaged together to calculate the ablation study from the right side of Figure 8.
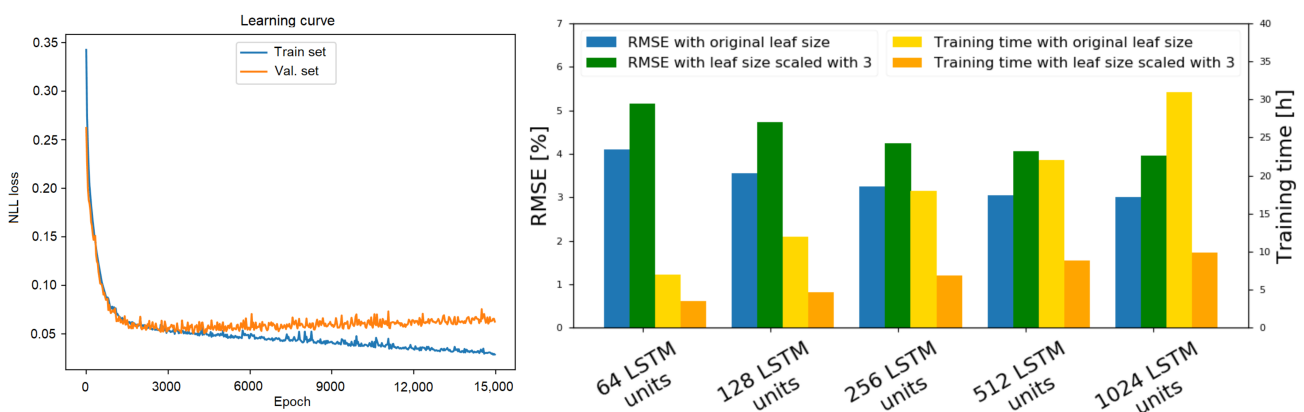

Figure 8. Learning curve and ablation of octree resolution and encoder-decoder model parameters. (**left side**) The evolution of the NLL loss on the training and validation set. (**right side**) Performance when training with different numbers of LSTM layers inside both the encoder and the decoder. We see that the RMSE percentages between the estimated and the ground truth driven trajectory decreases with respect to the added number of layers but gets capped after a certain threshold. Decreasing the resolution causes a small increase in RMSE but decreases the necessary training time in a significant manner.

The main takeaway from this study is that the performance of the system increases with respect to the number of added layers and neurons. However, the training time also increases at an exponential rate. We have concluded that the optimal structure for the deep network in Figure 1 is composed of 256 LSTM layers for the encoder and 256 LSTM layers for the decoder.

5.5. Deployment of OctoPath on the Nvidia AGX Xavier

The results of the network deployment on the Nvidia AGX Xavier mounted on RovisLab AMTU are presented in this section. The board supports three different power modes: 10 watts, 15 watts, and 30 watts, with each mode having several configurations with different CPU and GPU frequencies and a number of available cores. The comparison of OctoPath inference times when using different power modes are shown in Table 2. The *TensorRT* column represents the processing time using an optimized model with 4 bytes FP representation (UFF file with the TensorFlow operations replaced by TensorRT plugin nodes), while the *Native Tensorflow* column shows the processing time when using the original TF model (as ProtoBuf file.pb).

Table 2. Inference time measured on the NVIDIA AGX Xavier with different power mode (nvpmodel) and optimization level settings.

Nvidia AGX Xavier Power Mode	Number of Online Cores	CPU Maximal Frequency (MHz)	TensorRT (ms)	Native Tensorflow (ms)
MODE_10W	2	1200	41.24	314.66
MODE_15W	4	1200	29.89	207.12
MODE_30W_4CORE	4	1780	21.37	153.86
MODE_30W_6CORE	6	2100	17.85	121.38
MODE_MAXN	8	2265	14.23	89.61

Because of the input OcTree environment model, the network's memory requirements are very low. The native TF ProtoBuf .pb file is slightly less than 10 MB in size. OctoPath can be used in embedded systems with real-time performance, as the network will process a frame in 14.23 ms with the Max-N setup. To put it another way, the device is capable of generating 70 paths per second, which is more than enough for most applications. The importance of optimization combined with FP32, as well as the power mode, is emphasized when compared to native TensorFlow solutions. Our robot is more than capable of supplying the current required for the AGX Xavier to operate in MAX-N mode.

5.6. Discussion

In our experiments, the hybrid A* algorithm behaved better than the regression approach, mostly because of the structure of the octree environment model input data. This makes A* strictly dependent on the precision of the obstacle representation in the surrounding environment. Besides, the jittering effect of OctoPath may be a side effect of the decoder output's discrete nature. It will, however, provide a reliable ego-vehicle trajectory prediction over a given time horizon.

We intend to use the e-CAM130A synchronized quad cameras in future research to perform a full semantic segmentation on the received point cloud and to extend the validation of our approach to more use-cases. Learning-based approaches, we think, can deliver better results in the long run than conventional methods. This improvement would be achieved by training on more data, which would include a greater number of corner cases.

6. Conclusions

OctoPath, our self-supervised method for local trajectory prediction for autonomous vehicles within a finite horizon, is outlined in this paper. A discrete octree-based environment model with configurable resolution provides the input data, which is then fed

into a recurrent neural network encoder-decoder. To train the network, we used data collected with a LiDAR sensor mounted on our robot, RovisLab AMTU, a 1:4 scaled vehicle, and we performed simulation and indoor/outdoor navigation experiments. We base our evaluation against a hybrid A* algorithm, a regression-based approach, as well as against a CNN learning-based optimal path planning method, and we conclude that OctoPath is a valid candidate for local trajectory prediction in the autonomous control of mobile robots. Having a configurable environment resolution is an advantage especially on target edge devices, as seen from the deployment on the Nvidia AGX Xavier.

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

References

1. Xu, H.; Gao, Y.; Yu, F.; Darrell, T. End-to-end learning of driving models from large-scale video datasets. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2174–2182.
2. Jaritz, M.; 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.
3. Pendleton, S.D.; Andersen, H.; Du, X.; Shen, X.; Meghjani, M.; Eng, Y.H.; Rus, D.; Ang, M.H. Perception, planning, control, and coordination for autonomous vehicles. *Machines* **2017**, *5*, 6.
4. Grigorescu, S.M.; Trasnea, B.; Marina, L.; Vasilcoi, A.; Cocias, T. NeuroTrajectory: A Neuroevolutionary Approach to Local State Trajectory Learning for Autonomous Vehicles. *IEEE Robot. Autom. Lett.* **2019**, *4*, 3441–3448.
5. Bengio, Y.; Courville, A.; Vincent, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828.
6. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
7. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
8. González, D.; Pérez, J.; Milanés, V.; Nashashibi, F. A review of motion planning techniques for automated vehicles. *IEEE Trans. Intell. Transp. Syst.* **2015**, *17*, 1135–1145.
9. Grigorescu, S.; Trasnea, B.; Cocias, T.; Macesanu, G. A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **2020**, *37*, 362–386.
10. Amini, A.; Gilitschenski, I.; Phillips, J.; Moseyko, J.; Banerjee, R.; Karaman, S.; Rus, D. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robot. Autom. Lett.* **2020**, *5*, 1143–1150.
11. Pan, Y.; Cheng, C.A.; Saigol, K.; Lee, K.; Yan, X.; Theodorou, E.A.; Boots, B. Imitation learning for agile autonomous driving. *Int. J. Robot. Res.* **2020**, *39*, 286–302.
12. Bojarski, M.; Del Testa, 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.
13. Kahn, G.; Abbeel, P.; Levine, S. BADGR: An autonomous self-supervised learning-based navigation system. *arXiv* **2020**, arXiv:2002.05700.
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529.

15. Panov, A.I.; Yakovlev, K.S.; Suvorov, R. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia Comput. Sci.* **2018**, *123*, 347–353.

16. Wang, B.; Liu, Z.; Li, Q.; Prorok, A. Mobile Robot Path Planning in Dynamic Environments through Globally Guided Reinforcement Learning. *arXiv* **2020**, arXiv:2005.05420.

17. Salay, R.; Queiroz, R.; Czarnecki, K. *An Analysis of ISO 26262: Machine Learning and Safety in Automotive Software*; Technical Report, SAE Technical Paper; SAE: Pittsburgh, PA, USA, 2018.

18. Steffi, D.D.; Mehta, S.; Venkatesh, K.; Dasari, S.K. Robot Path Planning—Prediction: A Multidisciplinary Platform: A Survey. In *Data Science and Security*; Springer:Singapore 2021; pp. 211–219.

19. Cai, K.; Wang, C.; Cheng, J.; De Silva, C.W.; Meng, M.Q.H. Mobile Robot Path Planning in Dynamic Environments: A Survey. *arXiv* **2020**, arXiv:2006.14195.

20. Li, J.; Yang, S.X.; Xu, Z. A Survey on Robot Path Planning using Bio-inspired Algorithms. In Proceedings of the 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dali, China, 6–8 December 2019; pp. 2111–2116.

21. Wang, J.; Chi, W.; Li, C.; Wang, C.; Meng, M.Q.H. Neural RRT*: Learning-based optimal path planning. *IEEE Trans. Autom. Sci. Eng.* **2020**, *17*, 1748–1758.

22. Wu, H.; Chen, Z.; Sun, W.; Zheng, B.; Wang, W. Modeling trajectories with recurrent neural networks. In Proceedings of the 26th International Joint Conference on Artificial Intelligence IJCAI-17, Melbourne, Australia, 19–25 August 2017.

23. Park, S.H.; Kim, B.; Kang, C.M.; Chung, C.C.; Choi, J.W. Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 1672–1678.

24. Ma, Y.; Zhu, X.; Zhang, S.; Yang, R.; Wang, W.; Manocha, D. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. In Proceedings of the AAAI Conference on Artificial Intelligence 2019, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 6120–6127.

25. Altché, F.; de La Fortelle, A. An LSTM network for highway trajectory prediction. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 353–359.

26. Deo, N.; Trivedi, M.M. Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 1179–1184.

27. Kim, B.; Kang, C.M.; Kim, J.; Lee, S.H.; Chung, C.C.; Choi, J.W. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 399–404.

28. Hornung, A.; Wurm, K.M.; Bennewitz, M.; Stachniss, C.; Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Auton. Robot.* **2013**, *34*, 189–206.

29. Han, S. Towards efficient implementation of an octree for a large 3D point cloud. *Sensors* **2018**, *18*, 4398.

30. Vanneste, S.; Bellekens, B.; Weyn, M. 3DVFH+: Real-time three-dimensional obstacle avoidance using an Octomap. In Proceedings of the MORSE 2014—Model-Driven Robot Software Engineering, York, UK, 21 July 2014; number 1319, pp. 91–102.

31. Zhang, G.; Wu, B.; Xu, Y.L.; Ye, Y.D. Multi-granularity environment perception based on octree occupancy grid. *Multimed. Tools Appl.* **2020**, *79*, 26765–26785.

32. Wang, T.; Wu, Y.; Liang, J.; Han, C.; Chen, J.; Zhao, Q. Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. *Sensors* **2015**, *15*, 9681–9702.

33. Wong, J.; Chiang, C. A general theory for skid steering of tracked vehicles on firm ground. *Proc. Inst. Mech. Eng. Part D J. Automob. Eng.* **2001**, *215*, 343–355.

34. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.

35. Dolgov, D.; Thrun, S.; Montemerlo, M.; Diebel, J. Practical search techniques in path planning for autonomous driving. *Ann Arbor* **2008**, *1001*, 18–80.

36. Trasnea, B.; Marina, L.; Vasilcoi, A.; Pozna, C.; Grigorescu, S. GridSim: A Simulated Vehicle Kinematics Engine for Deep Neuroevolutionary Control in Autonomous Driving. In Proceedings of the 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 25–27 February 2019.