Speed Planning Based on Terrain-Aware Constraint Reinforcement Learning in Rugged Environments

Andong Yang, *Graduate Student Member, IEEE*, Wei Li, and Yu Hu, *Member, IEEE*

***Abstract*—Speed planning in rugged terrain poses challenges due to various constraints, such as traverse efficiency, dynamics, safety, and smoothness. This letter introduces a framework based on Constrained Reinforcement Learning (CRL) that considers all these constraints. In addition, extracting the terrain information as a constraint to be added to the CRL is also a barrier. In this letter, a terrain constraint extraction module is designed to quantify the semantic and geometric attributes of the terrain by estimating maximum safe speed. All networks are trained on simulators or datasets and eventually deployed on a real mobile robot. To continuously improve the planning performance and mitigate the error caused by the simulator-reality gap, we propose a feedback structure for detecting and preserving critical experiences during the testing process. The experiments in the simulator and the real robot demonstrate that our method can reduce the frequency of dangerous status by 45% and improve up to 71% smoothness.**

***Index Terms*—Speed planning, mobile robot, rugged environments, reinforcement learning.**

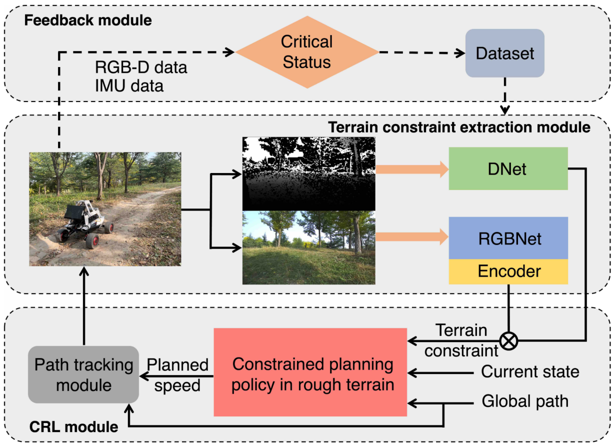

Fig. 1. Speed planning task in rugged environments needs to consider terrain constraint. This letter proposes a novel terrain constraint extraction module to extract terrain constraint using RGB-D data. A constrained reinforcement learning-based speed planning policy is proposed to find appropriate speed under various conditions and goals. We also design the feedback structure to continuously improve the accuracy of the networks and adjust hyperparameters by recording the experience from the testing phase.

I. Introduction

Speed planning for autonomous robots in rugged environments is challenging. In some typical scenarios, such as driving down a gravel slope, crossing a wood, and climbing over a barrier, incorrect speed can cause dangerous situations such as sliding, tipping, and colliding. Therefore, speed planning plays a critical role in ensuring robots are safe, especially on rugged terrain.

The speed planning task in rugged environments typically involves considering multiple constraints to achieve optimal performance. These constraints encompass preventing excessively low speeds, ensuring the planned speed is safe for the robot [1], avoiding deviations from the given path, and controlling changes in the vertical direction within a specified range to enhance sensor stability. Also, in rugged environments, terrain information is vitally important. The search-based method in [2] employs a visibility graph with speed and acceleration constraints to achieve the minimum time goal. However, this approach only processes fixed global constraints, not real-time and non-linear ones. Optimization-based [3] and switch-point [4] methods usually solve a set of equations to find speed, which allows complex non-linear constraints and considers system dynamics. But they are computationally expensive. In this letter, we propose a method based on Constrained Reinforcement Learning (CRL) to solve the speed planning problem on rugged terrain under multiple constraints.

To consider the influence of terrain in speed planning, existing methods utilize a range of sensors, such as LiDAR [5], RGB cameras [6], depth cameras [7], and others [8], to acquire and represent terrain information. Visual sensors are relatively inexpensive and can provide semantic and geometric information. Thus, we consider extracting terrain constraint based on an RGB-D camera in this letter. In recent visual-based methods, the experience-based approach [9] suggests speed based on the previous experience, which obtains the terrain constraint implicitly. However, the experience collection process requires the robot to drive on the same test path many times. Converting terrain constraint into traversability is another way. Such method [10] generally uses the end-to-end network to complete the conversion, which requires large amounts of data that is hard to collect in the real world to train the network. Higa et al.

Manuscript received 29 November 2023; accepted 2 January 2024. Date of publication 16 January 2024; date of current version 22 January 2024. This letter was recommended for publication by Associate Editor A. Zeng and Editor A. Faust upon evaluation of the reviewers' comments. This work was supported by the National Natural Science Foundation of China under Grants 62003323 and 62176250. *(Corresponding authors: Wei Li; Yu Hu.)*

The authors are with the Research Center for Intelligent Computing Systems, Institute of Computing Technology, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Beijing 100190, China (e-mail: yangandong19b@ict.ac.cn; liwei2019@ict.ac.cn; huyu@ict.ac.cn).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3354629, provided by the authors.
Digital Object Identifier 10.1109/LRA.2024.3354629

estimate robot energy consumption through the RGB-D data to reflect the difficulty of terrain traversed [11]. This method quantifies the terrain constraint and uses a neural network to learn it. However, in dangerous situations, like overturning, the energy consumption data is hard to collect, which means the network trained on this incomplete dataset may not be accurate. Also, the above methods feed RGB images and depth images into one network. This end-to-end structure might lead to performance degradation due to substantial variations between RGB and depth modalities [12]. In this letter, terrain constraint is designed to be quantified by the maximum speed at which the robot can safely travel in the corresponding terrain. We propose a visual-based terrain constraint extraction module consisting of two networks, as shown in Fig. 1. The RGBNet provides semantic understanding, while the DNet offers the maximum speed obtained from geometric information. By incorporating the terrain constraint into CRL, we propose the terrain-aware CRL.

All networks are trained on the RELLIS-3D dataset [13] and in the GAZEBO simulator and then deployed to an actual mobile robot. Since the training data may not cover all cases and the CRL method can interact with the environment in real-time, we propose a feedback structure that uses self-supervised learning to improve the accuracy of the terrain constraint extraction module further in the real world.

In this letter, we extend the capability of speed planning methods for mobile robots on rugged terrain. To summarize, our contributions are: (1) We represent the terrain information as a constraint and design a terrain constraint extraction module to extract it from RGB-D images. This module consists of two independent networks, effectively eliminating the performance degradation caused by substantial variations between RGB and depth modalities. (2) Terrain-aware CRL is introduced to solve the speed planning problem in rough terrain, which can handle the terrain constraint and all other nonlinear constraints. (3) We propose a self-supervised learning-based feedback structure to further improve the performance of the terrain constraint extraction module during the testing process.

II. Related Work

Speed planning under constraints aims to provide the path-tracking controller with suitable speed along the path so that the controller can provide angular speed to minimize the heading and lateral path-tracking errors [14]. Some representative methods can solve the minimum-time problems with speed and acceleration constraints [15] or other complex nonlinear constraints [16] while keeping safe. But those methods require high-definition maps, which are unavailable in our case. The end-to-end approach is another rapidly evolving approach to plan speed under constraints [17]. However, this method requires a lot of expert data.

Constrained reinforcement learning is a formulation to realize safe exploration and solve problems with complex nonlinear constraints [18]. CRL inherits the advantages of RL [19] while considering multiple constraints. Transforming the constraints to the reward function as a penalty item using lagrangian duality is one of the promising CRL methods [20]. But this method may violate constraints during training and lead to non-convex goals and constraints, which conducts a non-optimal solution. Other methods satisfy the constraints by restricting the actions that can be selected during exploration [21]. The safety layer method emerged to handle more complex continuous action constraints, which output constraint-compliant actions through neural networks [22]. However, the aforementioned method is generally used in simulators and designed for single constraint, while the speed planning task in this work involves multiple constraints. The latest method [23] alleviates the above problems and provides the possibility of practical application.

Extracting terrain constraint can be achieved via both proprioceptive and exteroceptive sensors. Chris J. Ostafew et al. determine terrain constraint by adding noise to the planned speed based on the experimental results in the previous round, continuously approaching the maximum speed of the current path [9], [14]. However, these methods can only be used on paths that have already been driven, and there may be risks during optimization. Arun Kumar Singh et al. use the coordinates between the robot wheels to confirm the maximum acceleration for current terrain [24]. However, this method can only perceive the current state and has a local optimization problem. There are also algorithms that use exteroceptive sensors [25], [26]. Shoya Higa et al. employ a neural network to estimate energy consumption from RGB-D images [11]. Mateus V. Gasparino et al. propose a traversability prediction neural network in a self-supervised manner based on visual data [7]. However, those methods only consider the semantic information of the terrain or directly assemble RGB and depth data as neural network inputs, which will worsen these cross-modal feature fusion as the networks go deeper due to the actual depth data being generally noisy [12].

III. Method

The overall framework of the proposed method is illustrated in Fig. 2, which consists of three major parts: terrain constraint extraction module, CRL module, and feedback module.

A. Visual-Based Terrain Constraint

To extract terrain constraint, we employ both geometric and semantic information. This module takes RGB-D data and outputs the terrain constraint represented by the maximum speed at which the robot can safely pass through the terrain. Specifically, we designed two networks, RGBNet f_{RGB} and DNet f_D. Dual network design decreases the error and training complexity caused by the pattern differences between multi-modality sensors. The networks can also be trained using different methods or datasets to improve accuracy.

The pixel classification information in images is one of the most useful semantic information, which can be obtained using the classification method for rugged environments [27]. However, current visual-based methods are not accurate enough to use directly in control. Some methods improve accuracy by adding several other types of sensor data like point cloud [28], thermal imagery [29] etc. But these sensors are expensive or

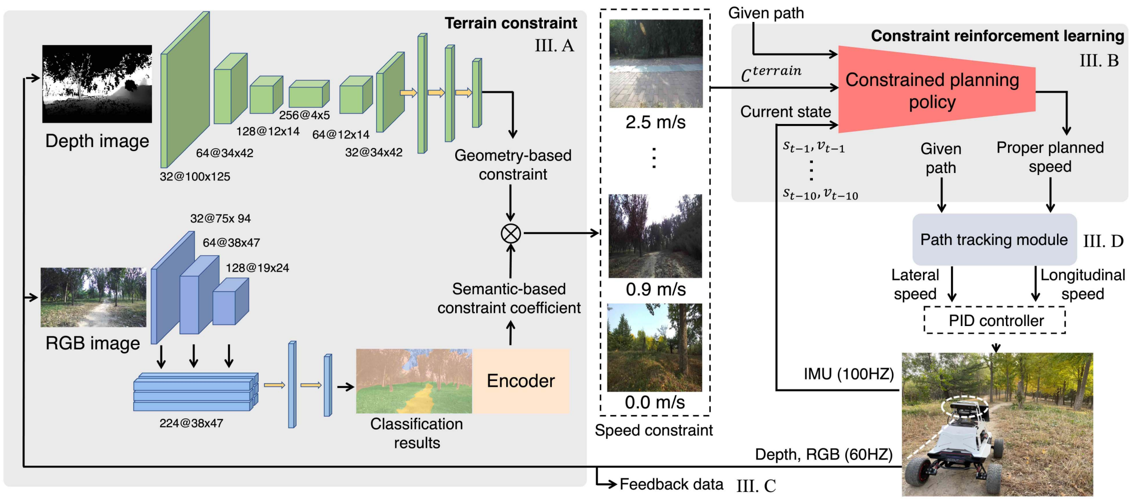

Fig. 2. Terrain constraint module extracts the geometric and semantic information, then fuses the information to obtain the speed constraint. The description under the neural network is $channel@height \times width$. The CRL-based policy will plan the final speed according to all constraints, goals, and global paths. The path-tracking module converts the planned speed into specific control commands.

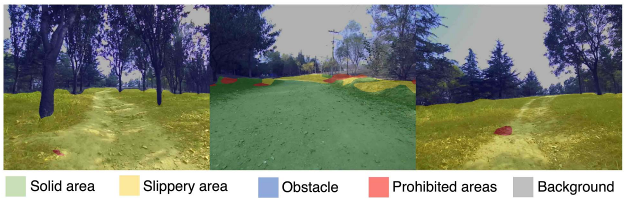

Fig. 3. Example of results for RGBNet.

TABLE I
TERRAIN CATEGORIES AND CORRESPONDING COEFFICIENTS

Terrain categories	Pixel-wise labels	Coefficient
Solid area	Concrete, Asphalt	1.0
Slippery area	Dirt, Rubble, Grass	0.58
Very slippery area	Puddle, Mud	0.23
Obstacle	Trees, Logs, person, barrier	0.0
Prohibited areas	Others	0.0
Background	Void, Sky	-

fragile. In addition, this classification information needs to be converted to terrain constraint properly. We propose several improvements to solve the problems above based on the existing image classification methods [27]. First, an encoder is added to convert semantic information to a semantic-based constraint, which is a coefficient in [0,1]. Second, the impact of classification errors in part of the image is mitigated by considering surrounding pixel information. Third, we introduce attention mechanisms to improve the rationality of results.

As shown in Fig. 2, we employ an image classification network RGBNet to obtain category information for each pixel. All RGB images $I^{RGB} \in \mathbb{R}^{3\times H\times W}$ are scaled to 375 pixels wide and 300 pixels high. The output of RGBNet is a probability mask $M^c \in \mathbb{R}^{1\times H\times W}$ that each value $M^c_i, i \in 1, 2, \ldots, H \times W$ represents the probability distribution that a pixel in the image belongs to a certain category of terrain $M^c = f_{RGB}(I^{RGB})$. This mask M represents the influence of different surface types on terrain constraint. Examples are shown in Fig. 3. Since the robot's travel path will be close to the given target path, pixels close to the global path should have more impact on speed. We first project the global path of the following 5 meters onto the depth map using IMU data. Then, the global path can be projected onto the RGB image using the correspondence between the depth and the RGB image. We define a mask $M^s \in \mathbb{R}^{1\times H\times W}$ to represent the influence of different areas of the image on terrain constraint. The 50 pixels around the path are defined as critical pixels P_c and the value of the corresponding positions in M^s are set to 1.2 and other pixels are sub-critical pixels P_s and are set to $(|P_c| + |P_s| - 1.2|P_c|)/|P_s|$. Especially, the pixels that fall into the sky category do not affect speed planning and, therefore, do not participate in subsequent calculations. So, we represent pixel positions that do not belong to the sky category as set A. Based on similar considerations, mask $M^d \in \mathbb{R}^{1\times H\times W}$ reflects the attribute that pixels closer to the mobile robot should have a more significant impact on speed. The position corresponding to the critical pixels in M^d linearly decreases from w based on the distance from the mobile robot, while $\sum_{i,j\in A} M^d_{i,j} = 1$.

All the above information is fed to an encoder:

$$C^{RGB}_t = \frac{1}{|A|} \sum_{i,j\ in\ A} V(M^c_{i,j}) \odot M^s_{i,j} \odot M^d_{i,j} \tag{1}$$

where $\odot$ is hadamard product, $C^{RGB}_t \in [0, 1]$ is the semantic-based constraint coefficient in time instance t. $V(M^c_{i,j})$ is a piecewise function, as shown in Table I. The process of obtaining parameters for $V(M^c_{i,j})$ is as follows. On the flat terrain of each road surface category, we gradually increase the linear speed of the robot until there is a rollover, skidding, excessive sensor jitter, or reaching the specified maximum linear speed of 2.5 m/s. Recording the final speed as the maximum line speed for the terrain of the current category. After several tests, the ratio of the recorded speed to the maximum linear speed is calculated, and the average value is taken as the function's parameter V. During this process, the robot's angular velocity is set to a maximum value of 0.5 rd/s. The RGBNet is trained using supervised learning on existing datasets RELLIS-3D [13].

The DNet f_D is used to extract the geometric information of terrain from the depth image. This work directly establishes a mapping of depth information to speed, thereby converting the geometric information of the terrain into a terrain constraint represented by the speed that can be used directly in subsequent planning algorithms. This mapping is implemented by the neural network DNet. The structure of DNet references the existing feature extraction network [30], which we believe has sufficient ability to extract depth features and map them to speeds. The upper left part in Fig. 2 shows DNet's architecture. The input is depth image $I^D \in \mathbb{R}^{1\times H\times W}$. The output is maximum safety speed which represents the geometry-based constraint of current terrain $C_t^D = f_D(I^D)$, where $C_t^D \in [0, 2.5]$. DNet can be trained using a supervised learning method based on the dataset collected in this letter. The DNet is trained in the GAZEBO simulator.

The final terrain constraint is the multiplication of the geometry-based terrain constraint and the semantic-based terrain constraint coefficients:

$$C_t^{terrain} = f_w(C_t^{RGB})C_t^D \tag{2}$$

where $C_t^{terrain} \in [0.0, 2.5]$, the weight of semantic information can be adjusted by the linear function $f_w(C_t^{RGB}) = \omega C_t^{RGB} + \varphi : [0, 1] \to [0, 1]$. In order to guarantee the discount belongs to [0,1], we set $f_w(1) = 1$. We initialize ω to 1 and φ to 0, which are then adjusted by the feedback structure.

B. Constrained Reinforcement Learning

The goal of speed planning is to generate a time-optimal speed profile on the given target path S^g under constraints. In this letter, the constraints include (1) the terrain constraint calculated in the previous module, (2) jerk-based stability constraint, which can reduce sudden changes in planned speed, thereby reducing vibration and improving sensor data quality, (3) robot dynamics, (4) safety constraints, (5) fixed constraints, such as speed range and acceleration range. We define discrete time $\{\Delta t, 2\Delta t, 3\Delta t, \ldots\}$ and using $t = \{1, 2, 3, \ldots\}$ to represent each moment.

Specifically, the optimal policy π aims to maximize the average linear speed v_t. Based on this goal, we define $G(v_t)$ in an iterative form so that each moment can be rewarded without waiting until the end of an experiment. In time instant t, $G(v_t) = \frac{1}{t}((t-1)G(v_{t-1}) + v_t)$ if $t > 1$ and $G(v_t) = v_t$ if $t = 1$, where v_t is current linear speed. Safety constraint $d_1(\boldsymbol{s_t})$ consists of three parts, where $\boldsymbol{s_t}$ is the current state of the robot containing position, orientation, speed, and acceleration a_t. The first part is the 3D orientation cannot exceed the specified range, such as the roll angle cannot exceed a certain value. The second part is the acceleration cannot exceed the specified safety range, and the third part is the speed cannot exceed terrain constraint speed $C_t^{terrain}$. A penalty $n\varepsilon_1$ will be set, where n is the number of the above parts that have been violated, and ε_1 is a hyperparameter. The path constraint $d_2(\boldsymbol{s_t}, S_t^g, t)$ comes from the planned speed that should be suitable for the robot to travel on a given target path, avoiding subsequent tracking algorithm failures due to inappropriate planned speed. We define this constraint as the average path tracking error to obtain real-time rewards $d_2(\boldsymbol{s_t}, S_t^g, t) = \frac{1}{t}(d_2(\boldsymbol{s_{t-1}}, S_{t-1}^g, t-1)(t-1) + ||S_t^g - \boldsymbol{s_t}||_2)$ if $t > 1$ and $d_2(\boldsymbol{s_t}, S_t^g, t) = ||S_t^g - \boldsymbol{s_t}||_2$ if $t = 1$. The jerk constraint $d_3(\boldsymbol{s_t})$ is directly defined as a piecewise function that gives a penalty ε_3 when it goes beyond the defined range, otherwise 0. Since dynamic models in rough terrain are challenging to model using traditional methods [31], we do not define or train the dynamic model directly; the constraint conducted by the dynamic model is implicit in the CRL network during training.

All goals and constraints of the speed planning problem constitute a CMDP problem [23]. From this, we further define the value functions of the constraints. We first define constraints set $\{d_i | i = 1, 2, \ldots, m\}$, where d_i is the cost of each constraint as defined above and m is the constraints number. Then the value function of each constraint is $V_\pi^{d_i}(\boldsymbol{s}) = E_\pi[\sum_{t=0}^{\infty} \gamma^t d_i | \boldsymbol{s_0} = \boldsymbol{s}]$, where γ is the discount factor, which indicates the speed at the farther moment has less impact on the present. Thus, the optimization problem can be defined as:

$$\begin{aligned}
\arg\max_{\pi_{\boldsymbol{\theta}}} \quad & E_{\boldsymbol{s}\sim\rho_0} E_{\pi_{\boldsymbol{\theta}}} \left[\sum_t^{\infty} \gamma^t G(v_t) | \boldsymbol{s_0} = \boldsymbol{s}\right] \\
\text{s.t} \quad & E_{\boldsymbol{s}\sim\rho_0}[V_{\pi_{\boldsymbol{\theta}}}^{d_i}(\boldsymbol{s})] \le e_i, i = 1, \ldots, m \\
& 0 \le v_t \le v_{\max} \\
& a_{\min} \le a_t \le a_{\max} \\
& t = 1, 2, \ldots
\end{aligned} \tag{3}$$

where e_i is the safety constraint bound for each constraint, ρ_0 is the starting state distribution, $a_{\min}$ and $a_{\max}$ are the maximum and minimum acceleration limits, respectively.

Inspired by [23], we solve this problem with CRL. The planning policy is a neural network with the parameter $\boldsymbol{\theta}$. The input includes current state $\boldsymbol{s}_t$, given target S^g, terrain constraint $C_t^{terrain}$. Based on the input, the value function is used to predict goal $G(v_t)$ and is a neural network with the parameter $\boldsymbol{\sigma_v}$. The constraint value function is also a neural network with the parameter $\boldsymbol{\sigma_c}$ and used to predict the sum of constraint $D_t = \sum_{i=1}^{m} \vartheta_i V_{\pi_{\boldsymbol{\theta}}}^{d_i}(\boldsymbol{s}_t)$, where ϑ_i is the normalization parameter that converts each constraint cost between 0 and 1. The process of updating the network is as follows. Collecting batch data of M episodes of horizon T in $B = \cup_{j=1}^{M} \cup_{t=1}^{T} \{(\boldsymbol{s}_t, S_t^g, \boldsymbol{a}_t, G_{t+1}, D_{t+1})_j\}$ according to current policy $\pi_{\boldsymbol{\theta}_k}$, where S_t^g is target state sequence of length N, $\boldsymbol{a}_t$ is planned speed (action). Improving policy with

$$\pi_{\boldsymbol{\theta}_{k'}} = \underset{\pi_{\boldsymbol{\theta}}}{argmax}\{L_R(\pi_{\boldsymbol{\theta}_k}, \pi_{\boldsymbol{\theta}}) - kl(\pi_{\boldsymbol{\theta}_k}, \pi_{\boldsymbol{\theta}})\} \tag{4a}$$

$$L_R(\pi_{\boldsymbol{\theta}_k}, \pi_{\boldsymbol{\theta}}) = \frac{1}{T}\sum_{t=1}^{T} \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t | \boldsymbol{s}_t, S_t^g)}{\pi_{\boldsymbol{\theta}_k}(\boldsymbol{a}_t | \boldsymbol{s}_t, S_t^g)} \hat{A}_t \tag{4b}$$

$$kl(\pi_{\boldsymbol{\theta}_k}, \pi_{\boldsymbol{\theta}}) = \alpha\sqrt{\frac{1}{T}\sum_{t=1}^{T} KL(\pi_{\boldsymbol{\theta}_k}, \pi_{\boldsymbol{\theta}})} \tag{4c}$$

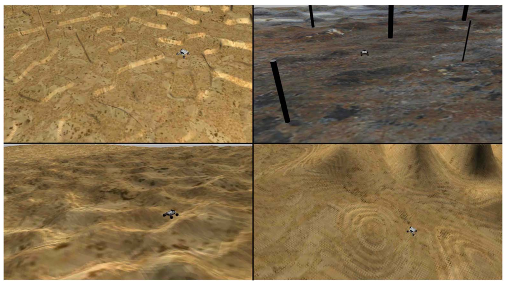

Fig. 4. Example of scenarios in the GAZEBO environment. We generated five maps, one of which inserted cylinders to simulate trees.

where k indicates the number of network updates, KL is Kullback-Leibler divergence, $\hat{A}_t$ is calculated based on value function network and GAE [32]. $\hat{A}_t$ can balance the variance and bias of the value function, which is widely used in the latest reinforcement learning algorithms. We employ a dual function to add constraints to the update of the policy. The dual function is obtained by the prime-dual method, and the network parameters $\pi_{\boldsymbol{\theta}_{k+1}}$ is updated with the goal of not violating constraints D based on $\pi_{\boldsymbol{\theta}_{k'}}$. When the policy network is updated, supervised learning is used to update the network parameters $\boldsymbol{\sigma}_v$ and $\boldsymbol{\sigma}_c$, respectively. This part is derived from mature algorithm [23] and is not described in detail. This module is trained in the GAZEBO simulator. The speed planned by best policy $\pi^*_{\boldsymbol{\theta}}$ will be fed into the path-tracking module to conduct specific control commands.

C. Feedback: Improving Performance With Experience

Since the sensor data in the simulator is less noisy than accurate sensor data, the DNet and some hyper-parameters trained in the simulator may be inaccurate in the real world. To ensure safety and allow the networks to learn from experience continuously, we propose a self-supervised learning-based feedback structure for detecting and preserving valuable experiences during the testing phase.

During testing, we deployed all algorithms on an actual mobile robot. If a critical status occurs, we will set the corresponding terrain constraint to 0 and added to the training dataset of DNet along with the robot state and sensor data. After each round of experiments, the DNet is trained with the extended dataset, and the hyperparameters are adjusted accordingly. The definition of critical status is defined as a scenario that exceeds the robot's capabilities or a dangerous condition, such as the roll angle or pitch angle of the mobile robot exceeding 30 degrees, an obstacle that exceeds the robot's ability to climb, or terrain that would cause the robot to overturn, etc. For this training process, we mainly refer to the standard self-supervised learning process [33]. Fig. 5 illustrates the feedback structure in this letter.

D. Path-Tracking Control

The path-tracking controller follows a series of N desired waypoints to control the mobile robot through a specific scene:

$$\mathcal{P}_{goal} := \{S^g_t | t \in \{1, 2, \ldots, N\}\} \qquad (5)$$

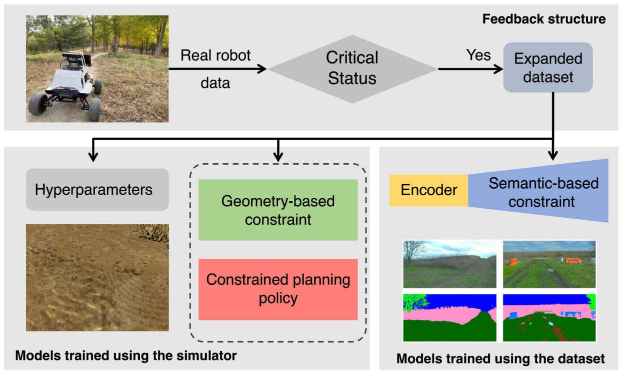
Feedback structure
Real robot data
Critical Status
Yes
Expanded dataset
Hyperparameters
Geometry-based constraint
Constrained planning policy
Encoder
Semantic-based constraint
Models trained using the simulator
Models trained using the dataset

Fig. 5. All networks are trained using the simulator or dataset. Then the feedback structure will improve the performance of each network. In addition, hyperparameters will also be adjusted based on the feedback data.

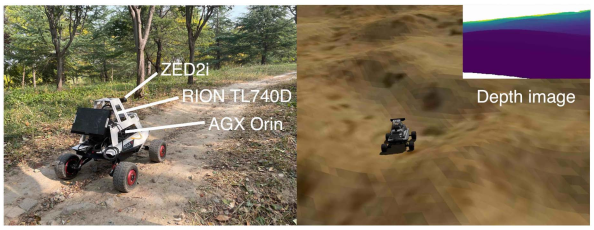

Fig. 6. Mobile robot and corresponding simulator.

each point S^g_i defining a translation (x, y, z), rotation $(roll, pitch, yaw)$. At time instance t, the inertial navigation system calculates an estimated pose of mobile robot $\boldsymbol{s}_t$ based on data from Inertial Measurement Unit (IMU), the path-tracking controller will set the target point as the nearest desired path vertex by Euclidean distance. Desired speed $\boldsymbol{a}^*_t \in \mathbb{R}$ will by generated for next S^g_t according to Section III-B. The path-tracking controller then computes a proper commanded speed using feedback linearization [34].

IV. Experiments

A. Hardware Setup

1) Real-World Setup: The platform used in this work is a custom-built differential steering mobile robot (weight 26 kg; LWH 0.612 m × 0.58 m × 0.295 m), as shown in Fig. 6 (left one). Sensors equipped on the robot include a stereo camera ZED2i and an IMU RION TL740D. The onboard computer is an NVIDIA AGX Orin. All sensors and algorithms run under the Robot Operating System (ROS), and the planning frequency is set to 10 Hz.

2) Simulation Setup: We opt for GAZEBO [35] for most training and simulation experiments based on a highly customized terrain. The mobile robot model used in GAZEBO is modeled 1:1 according to the mobile robot in the real world, as shown in Fig. 6 (right one). It provides RGB-D camera frames in the front to obtain surrounding environment information and IMU data. We built simulation environments and verified the proposed algorithm on one computer with an Intel Xeon E5-2650 v4 processor and Nvidia 2080Ti.

B. Model Training

The RGBNet is trained with a dataset collected in the real world. Specifically, the RGBNet network can use the existing rough terrain visual-based classification dataset like the RELLIS-3D dataset [13] to omit the dataset acquisition process. RELLIS-3D is a dataset collected in an off-road environment that contains 6,235 RGB images with 20 different pixel-wise coarse-grain labels. In this work, we unify all classes into five categories G, as shown in Table I. For each category, we calculate the binary mask $M_{G_i} \in \{0, 1\}^{H \times W}$, where $i = 0, 1, \ldots, G$. The training process of RGBNet is similar to the traditional classification network. We train the RGBNet with a cross-entropy loss L, comparing the output probability mask P_i with ground truth M_{G_i}:

$$\mathcal{L}_{ce} = -\sum_{h=1}^{H}\sum_{w=1}^{W}\sum_{i\in G} M_{G_i} log(M_i^c) \tag{6}$$

where $H = 300$, $W = 375$ is the dimensions of the image.

The DNet is trained using a dataset collected in the simulator. We first generate five elevation maps that contain three typical scenes: hillside, step, and hole with textures and physical properties to ensure the diversity of generated terrain [36]. Fig. 4 is an example of scenarios that import the elevation map to GAZEBO. The dataset consists of depth images I^D, and corresponding geometry-based terrain constraint C^{D^*}. In order to reduce the training difficulty and increase the generalization ability, we use the fixed speed $\{0, 0.5, 1.0, 1.5, 2.0, 2.5\}$ $\mathrm{m/s}$ as a geometry-based terrain constraint. The data collection process is as follows. We randomly set the vehicle position and select the control speed from 0 m/s to 2.5 m/s at an interval of 0.5 m/s. The robot travels until its status exceeds any of the constraints defined in Section III-B or there is no displacement within three seconds, which indicates the robot is in danger or stuck. Then the maximum speed is marked as the geometry-based terrain constraint C^{D^*} for the current image I^D. In addition, we randomly add Additive White Gaussian Noise (AWGN) with a standard deviation ranging from 10 to 30 to the depth images obtained from the simulator to improve the robustness. We collected 20000 sets of data in the simulator. The loss used to train the DNet is a Mean Squared Error (MSE):

$$\mathcal{L}_{MSE} = \frac{1}{B}\sum_{i=1}^{B}\left(C_i^D - C_i^{D^*}\right)^2 \tag{7}$$

where B is the batch size which we set to 1024.

The speed planning policy network is a three-layer perceptron with hidden nodes 256, 256, 256, and uses hyperbolic tangent as the activation function. This policy network uses reinforcement learning to learn strategies in interaction. Therefore, we use the GAZEBO simulator to train this network according to Section III-B. After deploying to the actual mobile robot, all parameters will be adjusted using the feedback data according to the description in Section III-C.

C. Comparing With the State-of-The-Art

We conducted experiments on a 2 kilometers long path containing rocks, grass, discolored leaves, mud, sand, and various representative geometric features in the real world. We compare our method with three widely used methods adapted to our mobile robot: model-based speed planning, learning-based end-to-end, and experience-based.

The model-based method can effectively learn robot dynamics and is widely used in robot planning and control tasks [37]. Therefore, we select a representative algorithm [38] as a comparison abbreviated from now on as MPC. This method predicts the future state based on the dynamics model of the robot and selects the speed corresponding to a better future state as a result. The learning-based end-to-end method is another representative planning approach, which directly establishes the mapping of RGB-D data to speed, and various constraints are reflected by the training data [25]. We used the same amount of simulator data to train our terrain constraint extraction module and this end-to-end network. We abbreviate this method as OneNet. The experience-based approach [9] infers terrain constraint from experience by repeating the same path more than 20 times. The planned speed of each round is obtained by adding disturbances to the previous round's planned speed based on the previous round's test results. This method can find the maximum speed on the path, but unsafe planning may occur during the search process, and the need to drive repeatedly on the same road is a significant drawback.

The following metrics are used to compare the navigation performance of our method with other methods.

- **Deviation**: The mean value of the path-tracking errors between the actual path and the given target path.
- **Dangerous**: The ratio of path points with dangerous situations like slide, collision, roll angle, or pitch angle of mobile robot exceeds 30 degrees which is the upper limit of the robot's capabilities in the entire path.
- **Smoothness**: The average speed change in all three-dimensional directions of space.
- **Success rate**: The calculation process is the same as [39]. Success is defined as not violating safety constraints and reaching the destination.

Metrics mean Intersection over Union (mIoU) and mean pixel accuracy (mAcc) illustrate the performance of the RGBNet and DNet. We also calculate the outlier ratio representing the percentage of image classification error exceeding 30% of the image area or the speed classification error exceeding 1.0 m/s to evaluate network stability. Table III shows the results.

In Table II, our method achieves better deviation, danger, and success rate metric results than SOTA approaches. The experience-based method achieved the highest average speed but a sharp decrease in safety and smoothness. Also, the experience-based method can only be used in paths that have been driven, and the speed planned during the optimization phase may not be safe. Therefore, although this method has the maximum average speed, its critical drawbacks make it difficult to apply widely. Fig. 7(a) shows that the trend is similar between the terrain constraint speed, the real speed, and the planned speed. So

TABLE II
BASELINE COMPARISONS AND ABLATION STUDY

Method	Baseline			Our				
	Model-based	OneNet	Experience-based	No DNet	No RGBNet	No terrain constraint	No CRL	Our
Deviation (m)	0.2±0.05	0.1±0.08	0.32±0.04	0.12±0.03	0.18±0.05	0.12±0.03	0.30±0.04	**0.1 ±0.03**
Dangerous (%)	10±1.1	7.3±1.5	5.6±1.1	6.6±1.3	6.2±1.2	10±1.5	8.0 ±1.7	**5.5±1.2**
Smoothness (m/s)	0.04± 0.08	0.09±0.012	0.15±0.09	0.05± 0.10	0.06± 0.09	**0.03±0.08**	0.14±0.11	0.04±0.007
Avg speed (m/s)	1.10±0.08	0.94±0.06	**1.61±0.04**	1.01±0.08	1.03±0.07	1.31±0.09	1.27±0.07	1.21±0.06
Succcess rate (%)	65.33±1.33	84±2.31	77.33±1.33	81.33±1.33	85.33±1.33	69.33± 2.31	61.33±1.33	**90.67±1.33**

TABLE III
TERRAIN CONSTRAINT EXTRACTION MODULE PERFORMANCE

Net	mIou	mAcc	Frame	Outlier
RGBNet	86.12	91.67	27	4.57%
DNet	-	-	32	3.42%

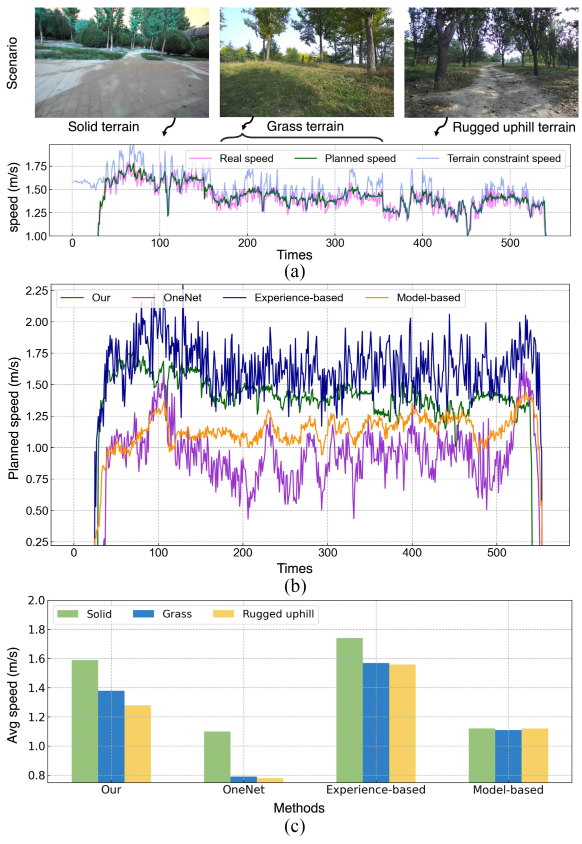

Fig. 7. (a) Terrain constraint speed, the planned speed and the real speed. (b) Planned speed of different methods in part of the complete experimental path. (c) The average planned speed of different methods in different terrains.

the change in speed is caused by algorithms rather than other reasons like tire slip. Fig. 7(b) shows the speed planned by different algorithms on different terrain. The experience-based method has the highest speeds but is highly volatile, which can significantly affect the subsequent tracking modules, leading to dangerous situations or an inability to track target paths. This unstable planning may stem from the lack of smoothness constraints. In contrast, the speed planned by our method is the most stable. Fig. 7(c) shows that the average speed planned by the Model-based method on the solid terrain is similar to that on

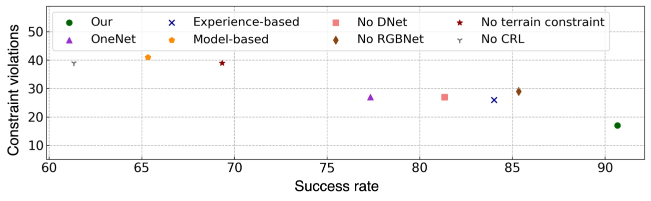

Fig. 8. Number of constraint violations and success rate for different methods.

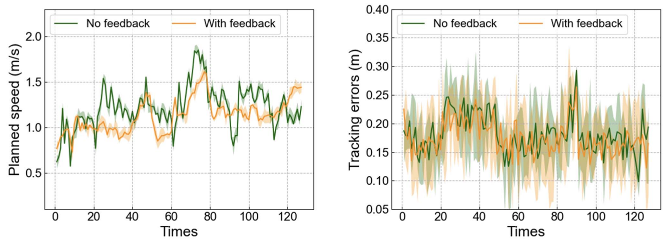

Fig. 9. Change of planned speed and tracking error after adding the feedback. The feedback module does not seriously affect planned speed and tracking errors but reduces the number of dangerous situations by 60%.

other terrains, which verifies that this method cannot distinguish the terrain characteristics. Meanwhile, the model-based method has the most dangerous speed. This proves that terrain constraint must be considered. Fig. 8 comprehensively evaluates different algorithms. The method that leans towards the bottom right corner is safer and can better complete planning tasks. It indicates that our method can best follow constraints and goals.

D. Ablation Study

We also conduct ablation experiments to evaluate each module in our method further. As shown in Table II, the methods with terrain constraint obtained from semantic or geometric information have less dangerous situations. This indicates that the terrain constraint based on fused information is better than using a single one. The method without terrain constraint has significantly more dangerous scenes. This indicates that the terrain constraint extraction module is necessary. The method without the CRL module uses the speed obtained from terrain constraint directly as the planned speed. The smoothness and deviation of this method become worse, indicating that the CRL module can consider various constraints and plan a smoother speed. Fig. 8 also shows that removing either module results in performance degradation.

In addition, we evaluate the feedback structure. High speed and safety are usually conflicting. When the dangerous situation of the planned speed decreases, it generally corresponds to a more conservative strategy: planning at a lower speed. As shown in Fig. 9. The proposed feedback module does not significantly slow the planned speed or affect path-tracking errors while reducing the dangerous situation by 60%. Therefore, the feedback structure proposed in this letter has a positive effect on improving performance in terms of safety.

V. Conclusion

This work proposes a speed planning policy based on CRL to plan appropriate speed based on various constraints. To quantify the terrain information, a novel terrain constraint extraction module is developed. In addition, we design a self-supervised learning-based feedback structure to improve the performance of networks further. The future work includes adding the mechanism for predicting future states based on current observation to improve the rationality of planning.

References

[1] W. Xu and J. M. Dolan, "Speed planning in dynamic environments over a fixed path for autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 3321–3327.

[2] J. Johnson and K. Hauser, "Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 2035–2041.

[3] Y. Zhang et al., "Speed planning for autonomous driving via convex optimization," in *Proc. IEEE 21st Int. Conf. Intell. Transp. Syst.*, 2018, pp. 1089–1094.

[4] Q. -C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Trans. Robot.*, vol. 30, no. 6, pp. 1533–1540, Dec. 2014.

[5] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments," *J. Field Robot.*, vol. 34, no. 5, pp. 940–984, 2017.

[6] A. Polevoy, C. Knuth, K. M. Popek, and K. D. Katyal, "Complex terrain navigation via model error prediction," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2022, pp. 9411–9417.

[7] C. Sevastopoulos and S. Konstantopoulos, "A survey of traversability estimation for mobile robots," *IEEE Access*, vol. 10, pp. 96331–96347, 2022.

[8] Z. Yu, S. Perera, H. Hauser, P. R. N. Childs, and T. Nanayakkara, "A tapered whisker-based physical reservoir computing system for mobile robot terrain identification in unstructured environments," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 3608–3615, Apr. 2022.

[9] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot, and J. Collier, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path tracking," *J. Field Robot.*, vol. 33, no. 1, pp. 133–152, 2016.

[10] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, "Learning ground traversability from simulations," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 1695–1702, Jul. 2018.

[11] S. Higa et al., "Vision-based estimation of driving energy for planetary rovers using deep learning and terramechanics," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3876–3883, Oct. 2019.

[12] X. Chen et al., "Bi-directional cross-modality feature propagation with separation-and-aggregation gate for RGB-D semantic segmentation," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 561–577.

[13] P. Jiang, P. Osteen, M. Wigness, and S. Saripalli, "RELLIS-3D dataset: Data, benchmarks and analysis," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 1110–1116.

[14] C. J. Ostafew, A. P. Schoellig, T. D. Barfoot, and J. Collier, "Speed daemon: Experience-based mobile robot speed scheduling," in *Proc. IEEE Can. Conf. Comput. Robot Vis.*, 2014, pp. 56–62.

[15] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, 1986.

[16] N. D. Van, M. Sualeh, D. Kim, and G.-W. Kim, "A hierarchical control system for autonomous driving towards urban challenges," *Appl. Sci.*, vol. 10, no. 10, 2020, Art. no. 3543.

[17] C. Lu, J. Gong, C. Lv, X. Chen, D. Cao, and Y. Chen, "A personalized behavior learning system for human-like longitudinal speed control of autonomous vehicles," *Sensors*, vol. 19, no. 17, 2019, Art. no. 3672.

[18] L. Brunke et al., "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 5, pp. 411–444, 2022.

[19] H. Lee, K. Kim, N. Kim, and S. W. Cha, "Energy efficient speed planning of electric vehicles for car-following scenario using model-based reinforcement learning," *Appl. Energy*, vol. 313, 2022, Art. no. 118460.

[20] H. Bharadhwaj, A. Kumar, N. Rhinehart, S. Levine, F. Shkurti, and A. Garg, "Conservative safety critics for exploration," in *Proc. 9th Int. Conf. Lear. Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=iaO86DUuKi

[21] M. Wen and U. Topcu, "Constrained cross-entropy method for safe reinforcement learning," *IEEE Trans. Autom. Control*, vol. 66, no. 7, pp. 3123–3137, 2021.

[22] G. Dulac-Arnold et al., "Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis," *Mach. Learn.*, vol. 110, no. 9, pp. 2419–2468, 2021.

[23] L. Yang et al., "Constrained update projection approach to safe policy optimization," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 9111–9124, 2022.

[24] A. K. Singh and K. M. Krishna, "Feasible acceleration count: A novel dynamic stability metric and its use in incremental motion planning on uneven terrain," *Robot. Auton. Syst.*, vol. 79, pp. 156–171, 2016.

[25] G. Kahn, P. Abbeel, and S. Levine, "BADGR: An autonomous self-supervised learning-based navigation system," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 1312–1319, Apr. 2021.

[26] X. Cai, M. Everett, J. Fink, and J. P. How, "Risk-aware off-road navigation via a learned speed distribution map," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 2931–2937.

[27] T. Guan, D. Kothandaraman, R. Chandra, A. J. Sathyamoorthy, K. Weerakoon, and D. Manocha, "GA-Nav: Efficient terrain segmentation for robot navigation in unstructured outdoor environments," *IEEE Robot. Automat. Lett.*, vol. 7, no. 3, pp. 8138–8145, Jul. 2022.

[28] A. Shaban, X. Meng, J. Lee, B. Boots, and D. Fox, "Semantic terrain classification for off-road autonomous driving," in *Proc. Conf. Robot Learn.*, 2022, pp. 619–629.

[29] Y. Iwashita, K. Nakashima, A. Stoica, and R. Kurazume, "TU-Net and TDeePLab: Deep learning-based terrain classification robust to illumination changes, combining visible and thermal imagery," in *Proc. IEEE Conf. Multimedia Inf. Process. Retrieval*, 2019, pp. 280–285.

[30] Y. Chen and T. D. Barfoot, "Self-supervised feature learning for long-term metric visual localization," *IEEE Robot. Automat. Lett.*, vol. 8, no. 2, pp. 472–479, Feb. 2023.

[31] A. Yang, W. Li, and Y. Hu, "SMS-MPC: Adversarial learning-based simultaneous prediction control with single model for mobile robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 10905–10912.

[32] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. 4th Int. Conf. Learn. Representations*, 2016. [Online]. Available: http://arxiv.org/abs/1506.02438

[33] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 4037–4058, Nov. 2021.

[34] C. Samson and K. Ait-Abderrahim, "Feedback control of a nonholonomic wheeled cart in cartesian space," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1991, pp. 1136–1137.

[35] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2004, vol. 3, pp. 2149–2154.

[36] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot.*, 2018, pp. 1–7.

[37] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 3, pp. 269–296, 2020.

[38] R. Lattarulo and J P. Rastelli, "A hybrid planning approach based on MPC and parametric curves for overtaking maneuvers," *Sensors*, vol. 21, no. 2, 2021, Art. no. 595.

[39] H. Karnan et al., "VI-IKD: High-speed accurate off-road navigation using learned visual-inertial inverse kinodynamics," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 3294–3301.