Applied Intelligence (2024) 54:8278–8295
https://doi.org/10.1007/s10489-024-05610-y

Efficient perception, planning, and control algorithm for vision-based automated vehicles

Der-Hau Lee[1]

Accepted: 10 June 2024 / Published online: 25 June 2024

Abstract

Autonomous vehicles have limited computational resources and thus require efficient control systems. The cost and size of sensors have limited the development of self-driving cars. To overcome these restrictions, this study proposes an efficient framework for the operation of vision-based automatic vehicles; the framework requires only a monocular camera and a few inexpensive radars. The proposed algorithm comprises a multi-task UNet (MTUNet) network for extracting image features and constrained iterative linear quadratic regulator (CILQR) and vision predictive control (VPC) modules for rapid motion planning and control. MTUNet is designed to simultaneously solve lane line segmentation, the ego vehicle's heading angle regression, road type classification, and traffic object detection tasks at approximately 40 FPS for 228×228 pixel RGB input images. The CILQR controllers then use the MTUNet outputs and radar data as inputs to produce driving commands for lateral and longitudinal vehicle guidance within only 1 ms. In particular, the VPC algorithm is included to reduce steering command latency to below actuator latency, preventing performance degradation during tight turns. The VPC algorithm uses road curvature data from MTUNet to estimate the appropriate correction for the current steering angle at a look-ahead point to adjust the turning amount. The inclusion of the VPC algorithm in a VPC-CILQR controller leads to higher performance on curvy roads than the use of CILQR alone. Our experiments demonstrate that the proposed autonomous driving system, which does not require high-definition maps, can be applied in current autonomous vehicles.

Keywords Automated vehicles · Deep neural network · Constrained iterative linear quadratic regulator · Model predictive control

1 Introduction

The use of deep neural network (DNN) techniques in intelligent vehicles has expedited the development of self-driving vehicles in research and industry. Self-driving cars can operate automatically because equipped perception, planning, and control modules operate cooperatively [1–3]. The most common perception components used in autonomous vehicles include cameras and radar/lidar devices; cameras are combined with DNN to recognize relevant objects, and radars/lidars are mainly used for distance measurement [2, 4]. Because of limitations related to sensor cost and size, current Active Driving Assistance Systems (ADASs) primarily rely on camera-based perception modules with supplementary radars [5].

✉ Der-Hau Lee
derhaulee@gmail.com

1 Department of Electrophysics, National Yang Ming Chiao Tung University, 1001 University Road, Hsinchu 300, Taiwan

To understand complex driving scenes, multi-task DNN (MTDNN) models that output multiple predictions simultaneously are often applied in autonomous vehicles to reduce inference time and device power consumption. In [6], street classification, vehicle detection, and road segmentation problems were solved using a single MultiNet model. In [7], the researchers trained an MTDNN to detect drivable areas and road classes for vehicle navigation. DLT-Net, presented in [8], is a unified neural network for the simultaneous detection of drivable areas, lane lines, and traffic objects. The network localizes the vehicle when a high-definition (HD) map is unavailable. The context tensors between subtask decoders in DLT-Net share mutual features learned from different tasks. A lightweight multi-task semantic attention network was proposed in [9] to achieve simultaneous object detection and semantic segmentation; this network boosts detection performance and reduces computational costs through the use of a semantic attention module. YOLOP [10] is a panoptic driving perception network that simultaneously performs traffic object detection, drivable area segmentation, and lane detection on an NVIDIA TITAN XP GPU at a speed of 41 FPS (frames per second). In the commercially available TESLA Autopilot system [11], images from cameras with different viewpoints are entered into separate MTDNNs that perform driving scene semantic segmentation, monocular depth estimation, and object detection tasks. The outputs of these MTDNNs are further fused in bird's-eye-view (BEV) networks to directly output a reconstructed aerial-view map of traffic objects, static infrastructure, and the road itself.

In a modular self-driving system, the environmental perception results can be sent to an optimization-based model predictive control (MPC) planner to generate spatiotemporal curves over a time horizon. The system then reactively selects optimal solutions over a short interval as control inputs to minimize the gap between target and current states [12]. These MPC models can be realized with various methods [e.g., active set, augmented Lagrangian, interior point, or sequential quadratic programming (SQP)] [13, 14] and are promising for vehicle optimal control problems. In [15], a linear MPC control model was proposed that addresses vehicle lane-keeping and obstacle avoidance problems by using lateral automation. In [16], an MPC control scheme combining longitudinal and lateral dynamics was designed for following velocity trajectories. Ref. [17] proposed a scale reduction method for reducing the online computational efforts of MPC controllers, and they applied it to longitudinal vehicle automation, achieving an average computational time of approximately 4 ms. In [14], a linear time-varying MPC scheme was proposed for lateral automobile trajectory optimization. The cycle time for the optimized trajectory to be communicated to the feedback controller was 10 ms. In addition, [18] investigated automatic weight determination for car-following control, and the corresponding linear MPC algorithm was implemented using CVXGEN [19], which solves the relevant problem within 1 ms.

The constrained iterative linear quadratic regulator (CILQR) method was proposed to solve online trajectory optimization problems with nonlinear system dynamics and general constraints [20, 21]. The CILQR algorithm constructed on the basis of differential dynamic programming (DDP) [22] is also an MPC method. The computational load of the well-established SQP solver is higher than that of DDP [24]. Thus, the CILQR solver outperforms the standard SQP approach in terms of computational efficiency; compared with the CILQR solver, the SQP approach requires a computation time that is 40.4 times longer per iteration [21]. However, previous CILQR-relates studies [20, 21, 23, 24] have focused on nonlinear Cartesian-frame motion planning. Alternatively, planning within the Frenét-frame can reduce problem dimensions because it enables vehicle dynamics to be solved in tangential and normal directions separately with the aid of road reference line [25]; furthermore, the corresponding linear dynamic equations [18, 26] do not have adverse effects when high-order Taylor expansion coefficients are truncated in the CILQR framework [cf. Section 2]. These considerations motivated us to use linear CILQR planners to control automated vehicles.

We proposed an MTDNN in [27] to directly perceive ego vehicle's heading angle (θ) and distance from the lane centerline (Δ) for autonomous driving. The vision-based MTDNN model in [27] essentially provides the information necessary for ego car navigation within Frenét coordinates without the need for HD maps. Nevertheless, this end-to-end autonomous driving approach performs poorly in environments that are not shown during the training phase [2]. In [28], we proposed an improved control algorithm based on a multi-task UNet architecture (MTUNet) that comprises lane line segmentation and pose estimation subnets. A Stanley controller [30] was then designed to control the lateral automation of an automobile. The Stanley controller takes θ and Δ yielding from the network as it's input for lane-centering [29]. The improved algorithm outperforms the model in [27] and has comparable performance to a multi-task-learning reinforcement-learning (MTL-RL) model [31], which integrates RL and deep-learning algorithms for autonomous driving. However, our algorithms presented in [28] have a variety of problems as follows. 1) Vehicle dynamic models are not considered in the Stanley controller, and the model has poor performance for lanes with rapid curvature changes [32]. 2) The proposed self-driving system does not consider road curvature, resulting in poor vehicle control on curvy roads [33]. 3) The corresponding DNN perception network lacks object detection capability, which is a core task in automated driving. 4) The DNN input has high dimensional resolution of 400 $\times$ 400, which results in long training and inference times.

To address these shortcomings, this paper proposes a new system for real-time automated driving based on the developments described in [28, 29]. First, a YOLOv4 detector [34] is added to the MTUNet for object detection. Second, the inference speed of MTUNet was increased by reducing the input size without sacrificing network performance. Third, a vision predictive control (VPC) algorithm is proposed for reducing the steering command delay by enabling steer correction at a look-ahead point by applying road curvature information. The VPC algorithm can also be combined with the lateral CILQR algorithm (denoted VPC-CILQR) to rapidly perform motion planning and automobile control. As shown in Fig. 1, the vehicle actuation latency (T_{act}) was shorter than the steering command latency (T_{lat}) in our simulation. This delay may

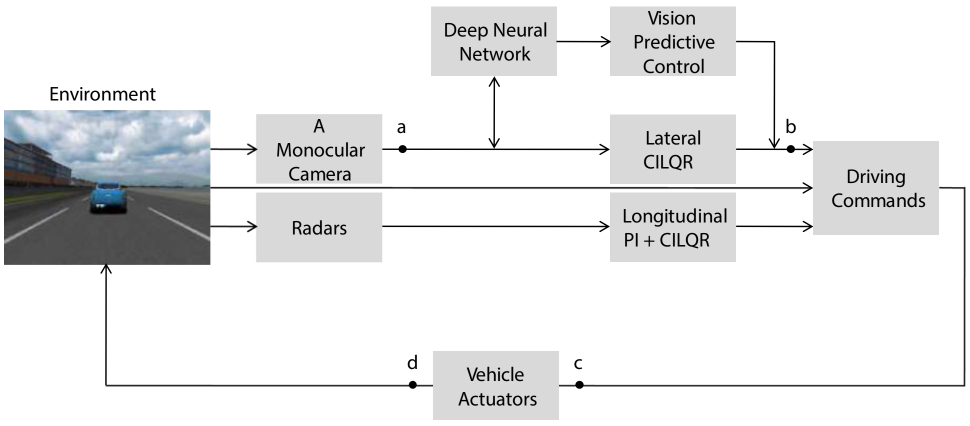

Fig. 1 Proposed vision-based automated driving framework. The system comprises the following modules: a multi-task DNN for perceiving surroundings, vision predictive control and CILQR controllers for vehicle motion planning and adherence to driving commands (steering, acceleration, and braking), and a PI controller combined with the longitudinal CILQR algorithm for velocity tracking. These modules receive input data from a monocular camera and a few inexpensive radars and operate collaboratively to operate the automated vehicle. The DNN, vision predictive control, and lateral and longitudinal CILQR algorithms are run efficiently every 24.52, 15.56, and 0.58 and 0.65 ms, respectively. In our simulation, the end-to-end latency from the camera output to the lateral controller output ($T_{a\rightarrow b} \equiv T_{lat}$) is longer than the actuator latency ($T_{c\rightarrow d} \equiv T_{act} = 6.66$ ms)

also be present in automated vehicles [35] or autonomous racing systems [36] and may induce instability in the system being controlled. Equipping the vehicle with low-level computers could further increase this steering command lag. Therefore, compensating algorithms such as VPC are key to cost-efficient automated vehicle systems.

In general, the research method of this paper is similar to those in [51, 52], which have also presented self-driving systems based on lane detection results. In [51]. an optimal LQR scheme with the sliding-mode approach was proposed to implement preview path tracking control for intelligent electric vehicles with optimal torque distribution between their motors. In [52], a safeguard-protected preview path tracking control algorithm was presented. The proposed preview control strategy comprises feedback and feedforward controllers for stabilizing tracking errors and preview control, respectively. The proposed controller was implemented and validated on open roads and Mcity, an automated vehicle platform. The tested vehicle was equipped with a commercial Mobileye module to detect lane markings.

The main goal of this work was to design a computationally efficient automated driving system for real-time lane-keeping and car-following. The contributions of this paper are as follows:

1. The proposed MTDNN scheme can execute simultaneous driving perception tasks at a speed of 40 FPS. The main difference between this scheme and previous MTDNN schemes is that the post-processing methods provide crucial parameters (lateral offset, road curvature, and heading angle) that improve local vehicular navigation.
2. The VPC-CILQR controller comprising the VPC algorithm and lateral CILQR solver is proposed to improve driverless vehicle path tracking. The method has a low online computational burden and can respond to steering commands in accordance with the concept of look-ahead distance.
3. We propose a vision-based framework comprising the aforementioned MTDNN scheme and CILQR-based controllers for operating an autonomous vehicle; the effectiveness of the proposed framework was demonstrated in challenging simulation environments without maps.

The remainder of this paper is organized as follows: the research methodology is presented in Section 2, the experimental setup is described in Section 3, and the results are presented and discussed in Section 4. Section 5 concludes this paper.

2 Methodology

The following section introduces each part of the proposed self-driving system. As depicted in Fig. 1, our system comprises several modules. The DNN is an MTUNet that can solve multiple perception problems simultaneously. The CILQR controllers receive data from the DNN and radars to compute driving commands for lateral and longitudinal motion planning. In the lateral direction, the lane line detection results from the DNN are input to the VPC module to compute steering angle corrections at a certain distance in front of the ego car. These corrections are then sent to the CILQR solver to predict a steering angle for the lane-keeping task. This two-step algorithm is denoted VPC-CILQR throughout the article. The other CILQR controller handles the car-following task in the longitudinal direction.

2.1 MTUNet network

As indicated in Fig. 2, the proposed MTDNN is a neural network with an MTUNet architecture featuring a common backbone encoder and three subnets for completing multiple tasks at the same time. The following sections describe each part.

2.1.1 Backbone and segmentation subnet

The shared backbone and segmentation (seg) subnet employ encoder-decoder UNet-based networks for pixel-level lane line classification task. Two classical UNets (UNet_2× [28] and UNet_1× [37]) and one enhanced version (MultiResUNet [37], denoted as MResUNet throughout the paper) were used to investigate the effects of model size and complexity on task performance. For UNet_2× and UNet_1×, each repeated block includes two convolutional (Conv) layers, and the first UNet has twice as many filters as the second. For MResUNet, each modified block consists of three 3 × 3 Conv layers and one 1 × 1 Conv layer. Table 1 summarizes the filter number and related kernel size of the Conv layers used in these models. The resulting total number of parameters of UNet_2×/UNet_1×/MResUNet is 31.04/7.77/7.26 M, and the corresponding total number of multiply accumulate operations (MACs) is 38.91/9.76/12.67 G. All 3 × 3 Conv layers are padded with one pixel to preserve the spatial resolution after the convolution operations are applied [38]. This setting reduces the network input size from 400 × 400

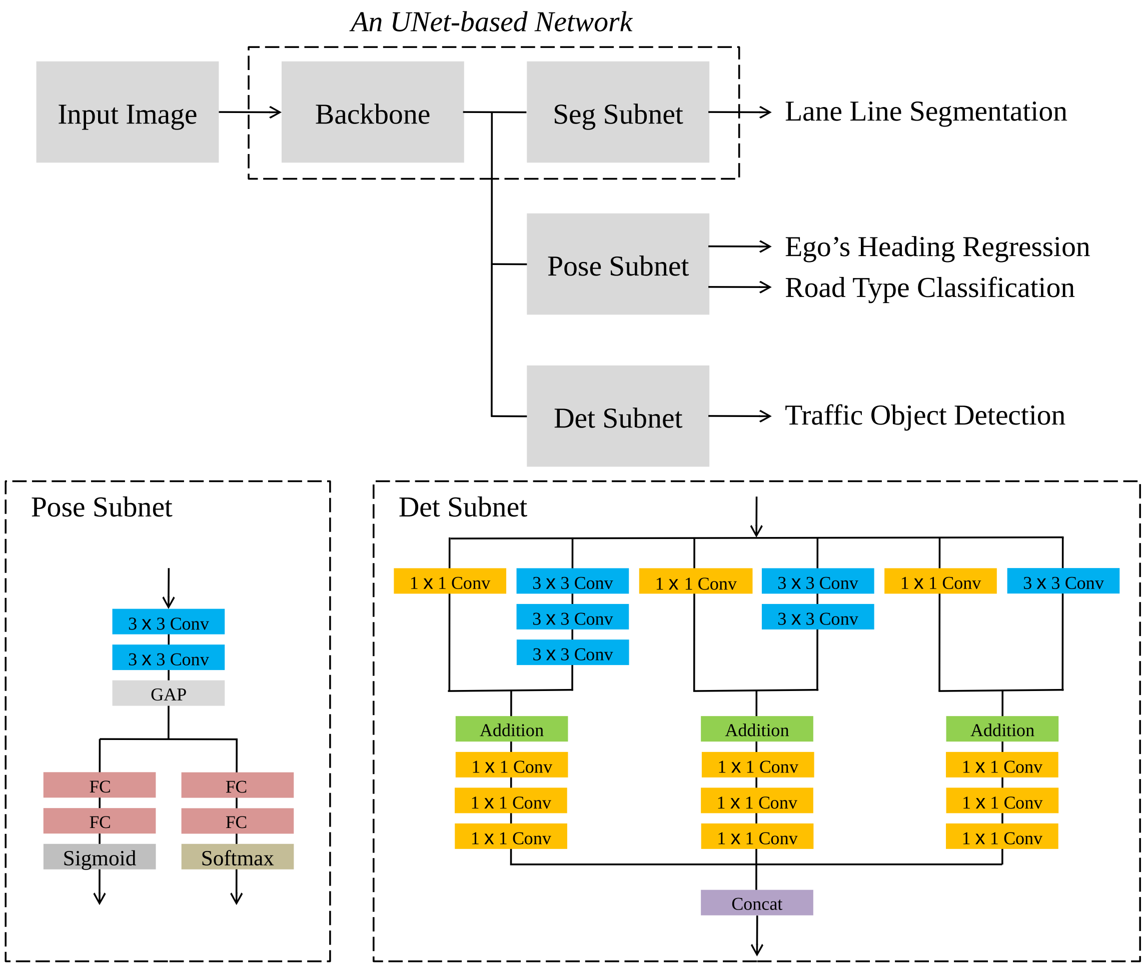

Fig. 2 Overview of proposed MTUNet architecture. The input RGB image of size 228 × 228 is fed into the model, which then performs lane line segmentation, ego vehicle's pose estimation, and traffic object detection at the same time. The backbone-seg-subnet is an UNet-based network; three variants of UNet (UNet_2× [28], UNet_1× [37], and MResUNet [37]) are compared in this work. The ReLU activation functions in pose and det subnets are not shown for simplicity

Table 1 Conv layers used in the UNet-based networks

Conv-block	UNet_2× [28]	UNet_1× [37]	MResUNet [37]
Block1[a]/9	64 (3 × 3)[b]	32 (3 × 3)	8 (3 × 3), 17 (3 × 3),
	64 (3 × 3)	32 (3 × 3)	26 (3 × 3), 51 (1 × 1)
Block2/8	128 (3 × 3)	64 (3 × 3)	17 (3 × 3), 35 (3 × 3),
	128 (3 × 3)	64 (3 × 3)	53 (3 × 3), 105 (1 × 1)
Block3/7	256 (3 × 3)	128 (3 × 3)	35 (3 × 3), 71 (3 × 3),
	256 (3 × 3)	128 (3 × 3)	106 (3 × 3), 212 (1 × 1)
Block4/6	512 (3 × 3)	256 (3 × 3)	71 (3 × 3), 142 (3 × 3),
	512 (3 × 3)	256 (3 × 3)	213 (3 × 3), 426 (1 × 1)
Block5	1024 (3 × 3)	512 (3 × 3)	142 (3 × 3), 284 (3 × 3),
	1024 (3 × 3)	512 (3 × 3)	427 (3 × 3), 853 (1 × 1)

[a] Block1 of UNet_2×/UNet_1× only contains one 3 × 3 Conv layer
[b] The notation n ($k \times k$) represents a Conv layer with n filters of kernel size $k \times k$

to 228 × 228 but preserves model performance and increases inference speed compared with the models in our previous work (the experimental results are presented in Section 4) [28]. That network used unpadded 3 × 3 Conv layers [39], and zero padding was therefore applied to the input to equalize the input–output resolutions [40]. In the training phase, the weighted cross-entropy loss is adopted to deal with the lane detection sample imbalance problem [41, 42] and is represented as

$$L_S = -\frac{N^-}{N^+ + N^-}\sum_{\tilde{y}=1}\log(\sigma(y)) - \frac{N^+}{N^+ + N^-}\sum_{\tilde{y}=0}\log(1-\sigma(y)), \tag{1}$$

where N^+ and N^- are the numbers of foreground and background samples in a batch of images, respectively; y is a predicted score; $\tilde{y}$ is the corresponding label; and σ is the sigmoid function.

2.1.2 Pose subnet

This subnet is mainly responsible for whole-image angle regression and road type classification problems, where the road involves three categories (left turn, straight, and right turn) designed to prevent the angle estimation from mode collapsing [28, 43]. The network architecture of the pose subnet is presented in Fig. 2; the pose subnet takes the fourth Conv-block output feature maps of the backbone as its input. Subsequently, the input maps are fed into shared parts including two consecutive Conv layers and one global average pooling (GAP) layer to extract general features. Lastly, the resulting vectors are passed separately through two fully connected (FC) layers before being mapped into a sigmoid/softmax activation layer for the regression/classification task. Table 2 summarizes the number of filters and output units of the corresponding Conv and FC layers, respectively. The expression MTUNet_2×/MTUNet_1×/MTMResUNet in Table 2 represents a multi-task UNet scheme in which subnets are built on the UNet_2×/UNet_1×/MResUNet model throughout the article. The pose task loss function, including L2 regression loss (L_R) and cross-entropy loss (L_C), is employed for network training; this function is represented as follows:

$$L_R = \frac{1}{2B}\sum_{i=1}^{B}\left|\sigma(\tilde{\theta}_i) - \sigma(\theta_i)\right|^2, \tag{2a}$$

$$L_C = -\frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{3}\tilde{p}_{ij}\log(p_{ij}), \tag{2b}$$

where $\tilde{\theta}$ and θ are the ground truth and estimated value, respectively; B is the input batch size; and $\tilde{p}$ and p are true and softmax estimation values, respectively.

2.1.3 Detection subnet

The detection (det) subnet takes advantage of a simplified YOLOv4 detector [34] for real-time traffic object (leading car) detection. This fully-convolutional subnet that has three branches for multi-scale detection takes the output feature maps of the backbone as its input, as illustrated in Fig. 2.

Table 2 Conv and FC layers used in the pose subnet of various MTUNets

Layer	MTUNet_2×	MTUNet_1× MTMResUNet
Conv1	1024 (3 × 3)	512 (3 × 3)
Conv2	1024 (3 × 3)	512 (3 × 3)
FC1-2	256, 256	256, 256
FC3-4	1, 3	1, 3

The initial part of each branch is composed of single or consecutive 3×3 filters for extracting contextual information at different scales [37], and a shortcut connection with one 1×1 filter from the input layer for residual mapping. The top of the addition layer contains sequential 1×1 filters for reducing the number of channels. The resulting feature maps of each branch have six channels (five for bounding box offset and confidence score predictions, and one for class probability estimation) with a size of $K = 15 \times 15$ to divide the input image into K grids. In this article, we select $M = 3$ anchor boxes, which are then shared between three branches according to the context size. Ultimately, spatial features from three detecting scales are concatenated together and sent to the output layer. Table 3 presents the design of detection subnet of MTUNets. The overall loss function for training comprises objectness (L_O), classification (L_{CL}), and complete intersection over union (CIoU) losses (L_{CI}) [44, 45]; these losses are constructed as follows:

$$\begin{aligned} L_O = &- \sum_{i=1}^{K\times M} I_i^o \left[\tilde{Q}_i \log(Q_i) + \left(1-\tilde{Q}_i\right)\log(1-Q_i)\right] \\ &- \lambda_n I_i^n \left[\tilde{Q}_i \log(Q_i) + \left(1-\tilde{Q}_i\right)\log(1-Q_i)\right], \end{aligned} \tag{3a}$$

$$\begin{aligned} L_{CL} = &- \sum_{i=1}^{K\times M} I_i^o \sum_{c\in classes} \tilde{p}_i(c)\log(p_i(c)) \\ &+ (1-\tilde{p}_i(c))\log(1-p_i(c)), \end{aligned} \tag{3b}$$

$$L_{CI} = 1 - IoU + \frac{E^2(\tilde{\mathbf{o}}, \mathbf{o})}{\beta^2} + \alpha\gamma, \tag{3c}$$

where $I_i^{o/n}$ = 1/0 or 0/1 indicates that the i-th predicted bounding box does or does not contain an object, respectively; $\tilde{Q}_i/Q_i$ and $\tilde{p}_i/p_i$ are the true/estimated objectness and class scores corresponding to each box, respectively; and λ_n is a hyperparameter intended for balancing positive and negative samples. With regard to CIoU loss, $\tilde{\mathbf{o}}$ and $\mathbf{o}$ are the central points of the prediction (B_p) and ground truth (B_{gt}) boxes, respectively; E is the related Euclidean distance; β is the diagonal distance of the smallest enclosing box covering B_p and B_{gt}; α is a tradeoff hyperparameter; and γ is used to measure aspect ratio consistency [44].

Table 3 Conv layers used in the detection subnet of various MTUNets

Layer	MTUNet_2×	MTUNet_1× MTMResUNet
Conv1-6	512 (3 × 3)	384 (3 × 3)
Conv7-9	512 (1 × 1)	384 (1 × 1)
Conv10-12	256 (1 × 1)	256 (1 × 1)
Conv13-15	256 (1 × 1)	256 (1 × 1)
Conv16-18	6 (1 × 1)	6 (1 × 1)

2.2 The CILQR algorithm

This section first briefly describes the concept behind CILQR and related approaches based on [20, 21, 46, 47]; it then presents the lateral/longitudinal CILQR control algorithm that takes the MTUNet inference and radar data as its inputs to yield driving decisions using linear dynamics.

2.2.1 Problem formulation

Provided a sequence of states $\mathbf{X} \equiv \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_N\}$ and the corresponding control sequence $\mathbf{U} \equiv \{\mathbf{u}_0, \mathbf{u}_1, ..., \mathbf{u}_{N-1}\}$ are within the preview horizon N, the system's discrete-time dynamics $\mathbf{f}$ are satisfied, with

$$\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{u}_i) \tag{4}$$

from time i to $i+1$. The total cost denoted by $\mathcal{J}$, including running costs $\mathcal{P}$ and the final cost $\mathcal{P}_f$, is presented as follows:

$$\mathcal{J}(\mathbf{x}_0, \mathbf{U}) = \sum_{i=0}^{N-1} \mathcal{P}(\mathbf{x}_i, \mathbf{u}_i) + \mathcal{P}_f(\mathbf{x}_N). \tag{5}$$

The optimal control sequence is then written as

$$\mathbf{U}^*\left(\mathbf{x}^*\right) \equiv \arg\min_{\mathbf{U}} \mathcal{J}(\mathbf{x}_0, \mathbf{U}) \tag{6}$$

with an optimal trajectory $\mathbf{x}^*$. The partial sum of $\mathcal{J}$ from any time step t to N is represented as

$$\mathcal{J}_t(\mathbf{x}, \mathbf{U}_t) = \sum_{i=t}^{N-1} \mathcal{P}(\mathbf{x}_i, \mathbf{u}_i) + \mathcal{P}_f(\mathbf{x}_N), \tag{7}$$

and the optimal value function $\mathcal{V}$ at time t starting at $\mathbf{x}$ takes the form

$$\mathcal{V}_t(\mathbf{x}) \equiv \arg\min_{\mathbf{U}_t} \mathcal{J}_t(\mathbf{x}, \mathbf{U}_t) \tag{8}$$

with the final time step value function $\mathcal{V}_N(\mathbf{x}) \equiv \mathcal{P}_f(\mathbf{x}_N)$.

In practice, the final step value function $\mathcal{V}_N(\mathbf{x})$ is obtained by executing a forward pass using the current control sequence. Subsequently, local control signal minimizations are performed in the proceeding backward pass using the following Bellman equation:

$$\mathcal{V}_i(\mathbf{x}) = \min_{\mathbf{u}} \left[\mathcal{P}(\mathbf{x}, \mathbf{u}) + \mathcal{V}_{i+1}(\mathbf{f}(\mathbf{x}, \mathbf{u}))\right]. \tag{9}$$

To compute the optimal trajectory, the perturbed function around the i-th state-control pair in (9) is used; this function is written as follows:

$$\mathcal{O}(\delta\mathbf{x},\delta\mathbf{u}) = \mathcal{P}_i(\mathbf{x}+\delta\mathbf{x},\mathbf{u}+\delta\mathbf{u}) - \mathcal{P}_i(\mathbf{x},\mathbf{u}) + \mathcal{V}_{i+1}(\mathbf{f}(\mathbf{x}+\delta\mathbf{x},\mathbf{u}+\delta\mathbf{u})) - \mathcal{V}_{i+1}(\mathbf{f}(\mathbf{x},\mathbf{u})). \tag{10}$$

This equation can be approximated to a quadratic function by employing a second-order Taylor expansion with the following coefficients:

$$\mathcal{O}_{\mathbf{x}} = \mathcal{P}_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^{\mathrm{T}}\mathcal{V}_{\mathbf{x}}, \tag{11a}$$
$$\mathcal{O}_{\mathbf{u}} = \mathcal{P}_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^{\mathrm{T}}\mathcal{V}_{\mathbf{x}}, \tag{11b}$$
$$\mathcal{O}_{\mathbf{xx}} = \mathcal{P}_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^{\mathrm{T}}\mathcal{V}_{\mathbf{xx}}\mathbf{f}_{\mathbf{x}} + \mathcal{V}_{\mathbf{x}}\mathbf{f}_{\mathbf{xx}}, \tag{11c}$$
$$\mathcal{O}_{\mathbf{ux}} = \mathcal{P}_{\mathbf{ux}} + \mathbf{f}_{\mathbf{u}}^{\mathrm{T}}\mathcal{V}_{\mathbf{xx}}\mathbf{f}_{\mathbf{x}} + \mathcal{V}_{\mathbf{x}}\mathbf{f}_{\mathbf{ux}}, \tag{11d}$$
$$\mathcal{O}_{\mathbf{uu}} = \mathcal{P}_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^{\mathrm{T}}\mathcal{V}_{\mathbf{xx}}\mathbf{f}_{\mathbf{u}} + \mathcal{V}_{\mathbf{x}}\mathbf{f}_{\mathbf{uu}}. \tag{11e}$$

The second-order coefficients of the system dynamics ($\mathbf{f}_{\mathbf{xx}}$, $\mathbf{f}_{\mathbf{ux}}$, and $\mathbf{f}_{\mathbf{uu}}$) are omitted to reduce computational effort [24, 46]. The values of these coefficients are zero for linear systems [e.g., Eq. (19) and Eq. (25)], leading to fast convergence in trajectory optimization.

The optimal control signal modification can be obtained by minimizing the quadratic $\mathcal{O}(\delta\mathbf{x},\delta\mathbf{u})$:

$$\delta\mathbf{u}^* = \arg\min_{\delta\mathbf{u}} \mathcal{O}(\delta\mathbf{x},\delta\mathbf{u}) = \mathbf{k} + \mathbf{K}\delta\mathbf{x}, \tag{12}$$

where

$$\mathbf{k} = -\mathcal{O}_{\mathbf{uu}}^{-1}\mathcal{O}_{\mathbf{u}}, \tag{13a}$$
$$\mathbf{K} = -\mathcal{O}_{\mathbf{uu}}^{-1}\mathcal{O}_{\mathbf{ux}} \tag{13b}$$

are optimal control gains. If the optimal control indicated in (12) is plugged into the approximated $\mathcal{O}(\delta\mathbf{x},\delta\mathbf{u})$ to recover the quadratic value function, the corresponding coefficients can be obtained [48]:

$$\mathcal{V}_{\mathbf{x}} = \mathcal{O}_{\mathbf{x}} - \mathbf{K}^{\mathrm{T}}\mathcal{O}_{\mathbf{uu}}\mathbf{k}, \tag{14a}$$
$$\mathcal{V}_{\mathbf{xx}} = \mathcal{O}_{\mathbf{xx}} - \mathbf{K}^{\mathrm{T}}\mathcal{O}_{\mathbf{uu}}\mathbf{K}. \tag{14b}$$

Control gains at each state ($\mathbf{k}_i$, $\mathbf{K}_i$) can then be estimated by recursively computing Eqs. (11), (13), and (14) in a backward process. Finally, the modified control and state sequences can be evaluated through a renewed forward pass:

$$\hat{\mathbf{u}}_i = \mathbf{u}_i + \lambda\mathbf{k}_i + \mathbf{K}_i\left(\hat{\mathbf{x}}_i - \mathbf{x}_i\right), \tag{15a}$$
$$\hat{\mathbf{x}}_{i+1} = \mathbf{f}\left(\hat{\mathbf{x}}_i, \hat{\mathbf{u}}_i\right), \tag{15b}$$

where $\hat{\mathbf{x}}_0 = \mathbf{x}_0$. Here λ is the backtracking parameter for line search; it is set to 1 in the beginning and designed to be reduced gradually in the forward-backward propagation loops until convergence is reached.

If the system has the constraint

$$\mathcal{C}(x,u) < 0, \tag{16}$$

which can be shaped using an exponential barrier function [20, 23]

$$\mathcal{B}(\mathcal{C}(x,u)) = q_1 \exp\left(q_2\mathcal{C}(x,u)\right) \tag{17}$$

or a logarithmic barrier function [21], then

$$\mathcal{B}(\mathcal{C}(x,u)) = -\frac{1}{t}\log\left(-\mathcal{C}(x,u)\right), \tag{18}$$

where q_1, q_2, and $t > 0$ are parameters. The barrier function can be added to the cost function as a penalty. Eq. (18) converges toward the ideal indicator function as t increases iteratively.

2.2.2 Lateral CILQR controller

The lateral vehicle dynamic model [26] is employed for steering control. The state variable and control input are defined as $\mathbf{x} = \left[\Delta\ \dot{\Delta}\ \theta\ \dot{\theta}\right]^{\mathrm{T}}$ and $\mathbf{u} = [\delta]$, respectively, where Δ is the lateral offset, θ is the angle between the ego vehicle's heading and the tangent of the road, and δ is the steering angle. As described in our previous work [28, 29], θ and Δ can be obtained from MTUNets and related post-processing methods, and it is assumed that $\dot{\Delta} = \dot{\theta} = 0$. The corresponding discrete-time dynamic model is written as follows:

$$\mathbf{x}_{t+1} \equiv \mathbf{f}(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t, \tag{19}$$

where

$$\mathbf{A} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & 0 & 0 \\ 0 & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ 0 & 0 & \alpha_{33} & \alpha_{34} \\ 0 & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ \beta_1 \\ 0 \\ \beta_2 \end{bmatrix},$$

with coefficients

$$\alpha_{11} = \alpha_{33} = 1, \quad \alpha_{12} = \alpha_{34} = dt,$$
$$\alpha_{22} = 1 - \frac{2(C_{\alpha f}+C_{\alpha r})dt}{mv}, \quad \alpha_{23} = \frac{2(C_{\alpha f}+C_{\alpha r})dt}{m},$$
$$\alpha_{24} = \frac{2(-C_{\alpha f}l_f+C_{\alpha r}l_r)dt}{mv}, \quad \alpha_{42} = \frac{2(C_{\alpha f}l_f-C_{\alpha r}l_r)dt}{I_z v},$$
$$\alpha_{43} = \frac{2(C_{\alpha f}l_f-C_{\alpha r}l_r)dt}{I_z}, \quad \alpha_{44} = 1 - \frac{2\left(C_{\alpha f}l_f^2-C_{\alpha r}l_r^2\right)dt}{I_z v},$$
$$\beta_1 = \frac{2C_{\alpha f}dt}{m}, \quad \beta_2 = \frac{2C_{\alpha f}l_f dt}{I_z}.$$

Here, v is the ego vehicle's current speed along the heading direction and dt is the sampling time. The model parameters for the experiments are as follows: vehicle mass m = 1150 kg, cornering stiffness $C_{\alpha f} = 80\,000$ N/rad, $C_{\alpha r} = 80\,000$ N/rad, center of gravity point $l_f = 1.27$ m, $l_r = 1.37$ m, and moment of inertia $I_z = 2000$ kgm^2.

The objective function ($\mathcal{J}$) containing the iterative linear quadratic regulator ($\mathcal{J}_{ILQR}$), barrier ($\mathcal{J}_b$), and end state cost ($\mathcal{J}_f$) terms can be represented as

$$\mathcal{J} = \mathcal{J}_{ILQR} + \mathcal{J}_b + \mathcal{J}_f, \tag{20a}$$

$$\mathcal{J}_{ILQR} = \sum_{i=0}^{N-1} (\mathbf{x}_i - \mathbf{x}_r)^{\mathrm{T}} \mathbf{Q} (\mathbf{x}_i - \mathbf{x}_r) + \mathbf{u}_i^{\mathrm{T}} \mathbf{R} \mathbf{u}_i, \tag{20b}$$

$$\mathcal{J}_b = \sum_{i=0}^{N-1} \mathcal{B}(u_i) + \mathcal{B}(\Delta_i), \tag{20c}$$

$$\mathcal{J}_f = (\mathbf{x}_N - \mathbf{x}_r)^{\mathrm{T}} \mathbf{Q} (\mathbf{x}_N - \mathbf{x}_r) + \mathcal{B}(\Delta_N). \tag{20d}$$

Here, the reference state $\mathbf{x}_r = \mathbf{0}$, $\mathbf{Q}/\mathbf{R}$ is the weighting matrix, and $\mathcal{B}(u_i)$ and $\mathcal{B}(\Delta_i)$ are the corresponding barrier functions:

$$\mathcal{B}(u_i) = -\frac{1}{t} \left(\log (u_i - \delta_{\min}) + \log (\delta_{\max} - u_i)\right), \tag{21a}$$

$$\mathcal{B}(\Delta_i) = \begin{cases} \exp(\Delta_i - \Delta_{i-1}) & \text{for} \quad \Delta_0 \geq 0, \\ \exp(\Delta_{i-1} - \Delta_i) & \text{for} \quad \Delta_0 < 0, \end{cases} \tag{21b}$$

where $\mathcal{B}(u_i)$ is used to limit control inputs and the high (low) steer bound is $\delta_{\max}$ ($\delta_{\min}$) $= \pi/6$ ($-\pi/6$) rad. The objective of $\mathcal{B}(\Delta_i)$ is to control the ego vehicle moving toward the lane center.

The first element of the optimal steering sequence is then selected to define the normalized steering command at a given time as follows:

$$\mathrm{SteerCmd} = \frac{\delta_0^*}{\pi/6}. \tag{22}$$

2.2.3 Longitudinal CILQR controller

In the longitudinal direction, a proportional-integral (PI) controller [49]

$$PI(v) = k_P e + k_I \sum_i e_i \tag{23}$$

is first applied to the ego car for tracking reference speed v_r under cruise conditions, where $e = v - v_r$ and k_P/k_I are the tracking error and the proportional/integral gain, respectively. The normalized acceleration command is then given as follows:

$$\mathrm{AcclCmd} = \tanh(PI(v)). \tag{24}$$

When a slower preceding vehicle is encountered, the AcclCmd must be updated to maintain a safe distance from that vehicle to avoid a collision; for this purpose, we use the following longitudinal CILQR algorithm.

The state variable and control input for longitudinal inter-vehicle dynamics are defined as $\mathbf{x}' = \begin{bmatrix} D & v & a \end{bmatrix}^{\mathrm{T}}$ and $\mathbf{u}' = [j]$, respectively, where a, $j = \dot{a}$, and D are the ego vehicle's acceleration, jerk, and distance to the preceding car, respectively. The corresponding discrete-time system model is written as

$$\mathbf{x}'_{t+1} \equiv \mathbf{f}'\left(\mathbf{x}'_t, \mathbf{u}'_t\right) = \mathbf{A}'\mathbf{x}'_t + \mathbf{B}'\mathbf{u}'_t + \mathbf{C}'\mathbf{w}', \tag{25}$$

where

$$\mathbf{A}' = \begin{bmatrix} 1 & -dt & -\frac{1}{2}dt^2 \\ 0 & 1 & dt \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B}' = \begin{bmatrix} 0 \\ 0 \\ dt \end{bmatrix},$$

$$\mathbf{C}' = \begin{bmatrix} 0 & dt & \frac{1}{2}dt^2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{w}' = \begin{bmatrix} 0 \\ v_l \\ a_l \end{bmatrix}.$$

Here, v_l/a_l is the preceding car's speed/acceleration, and $\mathbf{w}'$ is the measurable disturbance input [50]. The values of D and v_l are measured by the radar; v is known; and $a = a_l = 0$ is assumed. Here, MTUNets are used to recognize traffic objects, and the radar is responsible for providing precise distance measurements.

The objective function ($\mathcal{J}'$) for the longitudinal CILQR controller can be written as,

$$\mathcal{J}' = \mathcal{J}'_{ILQR} + \mathcal{J}'_b + \mathcal{J}'_f, \tag{26a}$$

$$\mathcal{J}'_{ILQR} = \sum_{i=0}^{N-1} \left(\mathbf{x}'_i - \mathbf{x}'_r\right)^{\mathrm{T}} \mathbf{Q}' \left(\mathbf{x}'_i - \mathbf{x}'_r\right) + \mathbf{u}'^{\mathrm{T}}_i \mathbf{R}' \mathbf{u}'_i, \tag{26b}$$

$$\mathcal{J}'_b = \sum_{i=0}^{N-1} \mathcal{B}'\left(u'_i\right) + \mathcal{B}'(D_i) + \mathcal{B}'(a_i), \tag{26c}$$

$$\mathcal{J}'_f = \left(\mathbf{x}'_N - \mathbf{x}'_r\right)^{\mathrm{T}} \mathbf{Q}' \left(\mathbf{x}'_N - \mathbf{x}'_r\right) + \mathcal{B}'(D_N) + \mathcal{B}'(a_N). \tag{26d}$$

Here, the reference state $\mathbf{x}'_r = \begin{bmatrix} D_r & v_l & a_l \end{bmatrix}$, and D_r is the reference distance for safety. $\mathbf{Q}'/\mathbf{R}'$ is the weighting matrix, and $\mathcal{B}'\left(u'_i\right)$, $\mathcal{B}'(D_i)$, and $\mathcal{B}'(a_i)$ are related barrier functions:

$$\mathcal{B}'\left(u'_i\right) = -\frac{1}{t'}\left(\log\left(u'_i - j_{\min}\right) + \log\left(j_{\max} - u'_i\right)\right), \tag{27a}$$

$$\mathcal{B}'(D_i) = \exp(D_r - D_i), \tag{27b}$$

$$\mathcal{B}'(a_i) = \exp(a_{\min} - a_i) + \exp(a_i - a_{\max}), \tag{27c}$$

where $\mathcal{B}'(D_i)$ is used for maintaining a safe distance, and $\mathcal{B}'(u'_i)$ and $\mathcal{B}'(a_i)$ are used to limit the ego vehicle's jerk and acceleration to $[-1, 1]$ m/s^3 and $[-5, 5]$ m/s^2, respectively.

The first element of the optimal jerk sequence is then chosen to update AccelCmd in the car-following scenario as

$$\text{AcclCmd} = \tanh\left(PI\left(v\right)\right) + j_0^*. \tag{28}$$

The brake command (BrakeCmd) gradually increases in value from 0 to 1 when D is smaller than a certain critical value during emergencies.

2.3 The VPC algorithm

The problematic scenario for the VPC algorithm is depicted in Fig. 3, which presents a top-down view of fitted lane lines produced using our previous method [29]. First, the detected line segments [Fig. 3(a)] were clustered using the density-based spatial clustering of applications with noise (DBSCAN) algorithm. Second, the resulting semantic lanes were transformed into BEV space by using a perspective transformation. Third, the least-squares quadratic polynomial fitting method was employed to produce parallel ego-lane lines [Fig. 3(b)]; either of the two polynomials can be represented as $y = f(x)$. Fourth, the road curvature κ was computed using the formula

$$\kappa = \frac{f''}{\left(1 + f'^2\right)^{3/2}}. \tag{29}$$

Because the curvature estimate from a single map is noisy, an average map obtained from eight consecutive frames was used for curve fitting. The resulting curvature estimates were then used to determine the correction value for the steering command in this study.

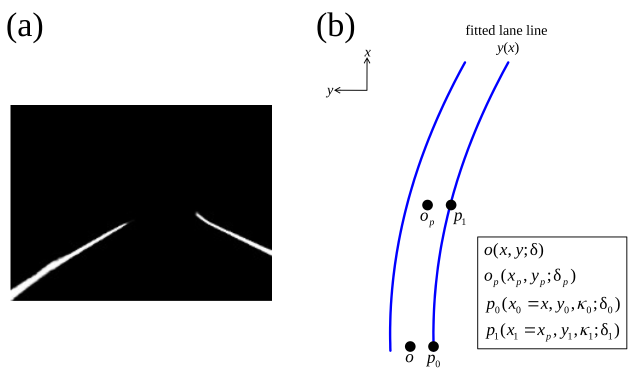

Fig. 3 Problematic scenario for the VPC algorithm. (a) An example DNN-output lane-line binary map at a given time in the egocentric view. (b) Aerial view of the fitted lane lines. Here, o is the current position of the ego vehicle and o_p is the look-ahead point, and p_0 and p_1 represent the corresponding lane points at the same x coordinates as o and o_p, respectively. κ and δ are the road curvature and steering angle of the ego vehicle, respectively. In this paper, the look-ahead distance $\overline{oo_p}$ = 10 m is used, which corresponds to a car speed of approximately 72 km/h [26]

As shown in Fig. 3(b), δ at o is the current steering angle. The desired steering angles at p_0 and p_1 can be computed using the local lane curvature [21]:

$$\delta_0 = \tan^{-1}\left(c\kappa_0\right), \tag{30a}$$
$$\delta_1 = \tan^{-1}\left(c\kappa_1\right), \tag{30b}$$

where c is an adjustable parameter. Hence, the predicted steering angle at a look-ahead point o_p can be represented as

$$\delta_p = \delta + (\delta_1 - \delta_0) \equiv \delta + \Delta\delta. \tag{31}$$

Compared with those in existing LQR-based preview control methods [51, 52], fewer tuning parameters are required when the VPC algorithm is included in the steering geometry model; moreover, the algorithm can be combined with other path-tracking models. For example, a VPC-CILQR controller can update the CILQR steering command [Eq. (22)] as follows:

$$\text{VPC\_SteerCmd} = \begin{cases} \text{SteerCmd} + |\Delta\delta| & \text{if} \quad \text{SteerCmd} \geq 0, \\ \text{SteerCmd} - |\Delta\delta| & \text{if} \quad \text{SteerCmd} < 0. \end{cases} \tag{32}$$

In summary, the proposed VPC algorithm uses the future road curvature at a look-ahead point 10 m in front of the ego car (Fig. 3) as input to generate the updated steering inputs. This algorithm is applied before the ego car enters a curvy road to improve tracking performance. Accurate and complete future road shape prediction is crucial for developing preview path-tracking control algorithms [52]. However, whether the necessary information can be obtained is greatly dependent on the maximum perception range of lane detection modules. As demonstrated in Fig. 5, LLAMAS [57] data are more useful than TORCS [28] or CULane [56] datasets for developing algorithms with such path-tracking functionality. A nonlinear MPC approach using high-quality predicted lane curvature data can achieve better control performance over the proposed method; however, if computational cost is a concern, such a nonlinear approach may not necessarily be preferred. The following sections describe validation experiments where the proposed algorithm was compared against other control algorithms.

3 Experimental setup

The proposed MTUNets extract local and global contexts from input images to simultaneously perform segmentation, detection, and pose tasks. Because the these tasks have different learning rates [40, 53, 54], the proposed MTUNets were trained in a stepwise instead of end-to-end manner to help the backbone network learn common features. The training strategy, image data, and validation are described as follows.

Table 4 Datasets used in the experiments

Dataset	Scenarios	No. of images	Labels	No. of traffic objects	Sources
CULane	urban, highway	28368	ego-lane lines, bounding boxes	80437	[56], this work
LLAMAS	highway	22714		29442	[57], this work
TORCS		42747	ego-lane lines, bounding boxes, ego's heading, road type	30189	[28]

3.1 Network training strategy

The MTUNets were trained in three stages. The pose subnet was first trained through stochastic gradient descent (SGD) with a batch size (bs) of 20, momentum (mo) of 0.9, and learning rate (lr) starting from 10^{-2} and decreasing by a factor of 0.9 every 5 epochs for a total of 100 epochs. The detection and pose subnets were then trained jointly with the parameters obtained in the first training stage and using the SGD optimizer with bs = 4, mo = 0.9, and lr = 10^{-3}, 10^{-4}, and 10^{-5} for the first 60 epochs, the 61st to 80th epochs, and the last 20 epochs, respectively. All subnets (detection, pose, and segmentation) were trained together in the last stage with the pretrained model obtained in the previous stage and using the Adam optimizer. Bs and mo were set to 1 and 0.9, respectively, and lr was set to 10^{-4} for the first 75 epochs and 10^{-5} for the last 25 epochs. The total loss in each stage was a weighted sum of the corresponding losses [55].

3.2 Image datasets

We conducted experiments on the artificial TORCS [28] and real-world CULane [56] and LLAMAS [57] datasets. The summary statistics of the datasets are presented in Table 4. The customized TORCS dataset has joint labels for all tasks, whereas the original CULane/LLAMAS dataset only contained lane line labels. Thus, we annotated each CULane and LLAMAS image with traffic object bounding boxes to mimic the TORCS dataset. Correspondingly, the TORCS, CULane, and LLAMAS datasets had approximately 30 K, 80 K, and 29 K labeled traffic objects, respectively. To determine anchor boxes for the detection task, the *k*-means algorithm [58] was applied to partition the ground truth boxes. The CULane and LLAMAS datasets lack ego vehicle angle labels; therefore, these datasets could only be used for evaluations in segmentation and detection tasks. The ratio of the number of images used in the training phase to that used in the test phase was approximately 10 for all datasets, as in our previous works [28, 29]. Recall/average precision (AP; IoU was set to 0.5), recall/F1 score, and accuracy/mean absolute error (MAE) were used to evaluate model performance in detection, segmentation, and pose tasks, respectively.

3.3 Autonomous driving simulation

The open-source driving environment TORCS provides sophisticated physics and graphics engines; it is therefore ideal for not only visual processing but also vehicle dynamics research [59]. The ego vehicle controlled by our self-driving framework was driven autonomously on unseen TORCS roads [e.g., Tracks A and B in Fig. 4] to validate the effectiveness of our approach. All experiments, including both MTUNet training and testing and driving simulations, were conducted on a PC equipped with an INTEL i9-9900K CPU, 64 GB of RAM, and an NVIDIA RTX 2080 Ti GPU with 4352 CUDA cores and 11 GB of GDDR memory. The control frequency for the ego vehicle in TORCS was approximately 150 Hz on this computer.

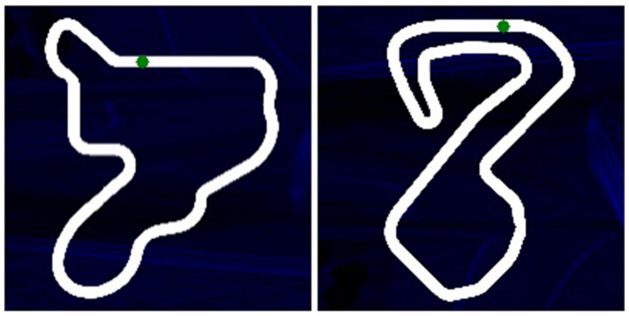

Fig. 4 Tracks A (left) and B (right) for dynamically evaluating proposed MTUNet and control models. The total length of Track A/B (Track 7/8 in [28]) was 2843/3919 m with lane width 4 m, and the maximum curvature was approximately 0.03/0.05 1/m, which was curvier than a typical road [60]. The self-driving car drove in a counterclockwise direction, and the starting locations are marked by green filled circle symbols. A self-driving vehicle [31] could not finish a lap on Track A using the direct perception approach [61]

Table 5 Performance of trained MTUNets on the test data

Network	Dataset	Task config.	Det		Seg		Pose	
			Recall	AP (%)	Recall	F1 score	Heading MAE (rad)	Road type accuracy (%)
MTUNet_2×	CULane	Det + Seg	0.858	65.32	0.694	0.688	–	–
MTUNet_1×			0.831	58.67	0.658	0.595	–	–
MTMResUNet			0.852	54.28	0.559	0.568	–	–
MTUNet_2×	LLAMAS	Det + Seg	0.942	64.42	0.935	0.827	–	–
MTUNet_1×			0.946	59.96	0.936	0.831	–	–
MTMResUNet			0.950	57.40	0.736	0.748	–	–
MTUNet_2×	TORCS	Pose	–	–	–	–	0.004	90.42
MTUNet_1×			–	–	–	–	0.005	90.48
MTMResUNet			–	–	–	–	0.006	83.77
MTUNet_2×		Det + Seg	0.976	71.51	0.905	0.889	–	–
MTUNet_1×			0.974	66.14	0.904	0.894	–	–
MTMResUNet			0.968	66.12	0.833	0.869	–	–
MTUNet_2×		Det + Seg + Pose	0.952	65.83	0.922	0.883	0.005	87.08
MTUNet_1×			0.956	59.25	0.901	0.882	0.004	94.30
MTMResUNet			0.959	51.88	0.830	0.855	0.007	80.46

4 Results and discussions

Table 5 presents the performance results of the MTUNet models on the testing data for various tasks. Table 6 lists the number of parameters, computational complexity, and inference speed of each scheme as a comparison of computational efficiency. As described in Section 2, although the input size of the MTUNet models was reduced by the use of padded 3 × 3 Conv layers, model performance was not affected; MTUNet_2×/MTUNet_1× achieved similar results to our previous model in the segmentation and pose tasks on the TORCS and LLAMAS datasets [28]. For complex CULane data, the MTUNet model performance performed worse than the SCNN [56], the original state-of-the-art method for this dataset; however, the SCNN had lower inference speed because of its higher computational complexity [10]. The MTUNet models are designed for real-time control of self-driving vehicles; the SCNN model is not. Of the three considered MTUNet variants, MTUNet_2× and MTUNet_1× outperformed MTMResUNet on all datasets if each model jointly performed the detection and segmentation tasks (first, second, and fourth row of Table 5). This result differs from that of a previous study on a single-segmentation task for biomedical images [37]. Task gradient interference can reduce the performance of an MTDNN [62, 63]; in this case, the MTUNet_2× and MTUNet_1× models outperformed the complex MTMResUNet network because of their elegant architecture. When the pose task was included (last row of Table 5), MTUNet_2× and MTUNet_1× also outperformed MTMResUNet on all evaluation metrics; the decreasing AP scores for the detection task are attributable to an increase in false positive (FP) detections. However, for all models, the inclusion of the pose task only decreased the recall scores for the detection task by approximately 0.02 (last two rows of Table 5); nearly 95% of the ground truth boxes were still detected during when the models simultaneously performed all tasks. Following the method for efficiency analysis used in [64] (Sec. V. B. in [64]), this study computed the densities of the detection AP and road type accuracy scores using the data in the last row of Tables 5/6. MTUNet_1× had higher efficiency in terms of parameter utilization than did MTUNet_2×. MTUNet_1× was 3.26 times smaller than

Table 6 Results for MTUNets in terms of parameters (Params), MACs, and FPS

Network	Task config.	Params	MACs	FPS
MTUNet_2×	Pose	19.37 M	13.93 G	74.02
MTUNet_1×		4.98 M	3.51 G	122.72
MTMResUNet		5.37 M	2.70 G	99.08
MTUNet_2×	Det + Seg	68.62 M	47.36 G	24.44
MTUNet_1×		21.70 M	12.89 G	43.58
MTMResUNet		21.97 M	15.98 G	27.79
MTUNet_2×	Det + Seg + Pose	83.31 M	50.55 G	23.28
MTUNet_1×		25.50 M	13.69 G	40.77
MTMResUNet		26.56 M	16.95 G	27.30

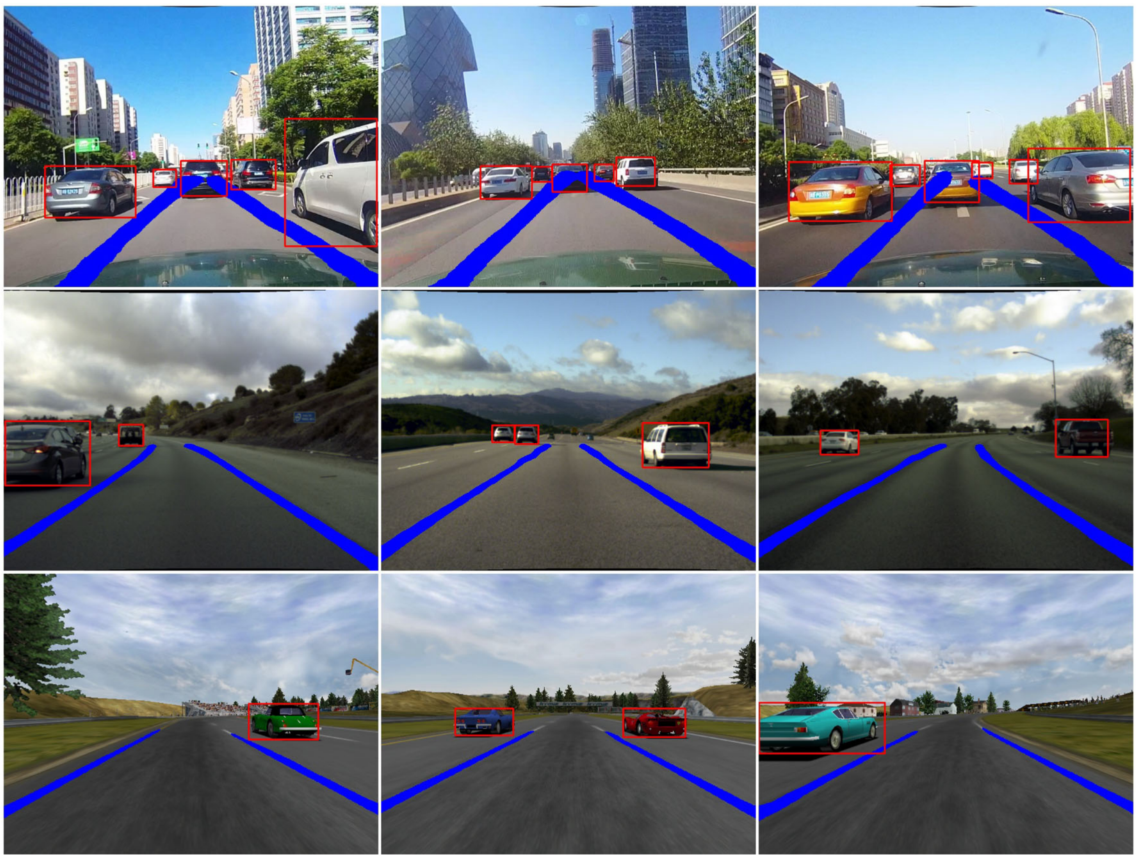

Fig. 5 Example traffic object and lane-line detection results for the MTUNet_1× network on CULane (first row), LLAMAS (second row), and TORCS (third row) images

MTUNet_2× and achieved a 1.75 times faster inference speed (40.77 FPS); this speed is comparable to that of the YOLOP model [10]. These results indicate that MTUNet_1× is the most efficient model for collaborating with controllers to achieve automated driving. The MTUNet_1× model can also be run on a low-performance computer with only a few gigabytes of GDDR memory. For a computer with a GTX 1050 Max-Q GPU with 640 CUDA cores and 4 GB of GDDR memory, the MTUNet_1× model achieved an inference speed of 14.69 FPS for multi-task prediction. Figure 5 presents example MTUNet_1× network outputs for both traffic objects and lane detection on all datasets.

To objectively evaluate the dynamic performance of the autonomous driving algorithms, lane-keeping and car-following maneuvers were performed on the challenging tracks, Tracks A and B, as shown in Fig. 4. The SQP-based controllers were implemented using the ACADO toolkit [65] for comparison with the CILQR-based controllers. All settings for these algorithms were the same as summarized in Table 7. For the lateral control experiments, autonomous vehicles were designed to drive at various cruise speeds on Tracks A and B. The θ and Δ results for the VPC-CILQR, CILQR, VPC-SQP, and SQP algorithms are presented in Figs. 6, 7, 8, 9, 10, 11, 12, and 13. The results of the CILQR and SQP controllers for the longitudinal control experiments are presented in Fig. 14. The MAEs for θ, Δ, v, and D in Figs. 6–14 are listed in Table 8. Table 9 presents the average time to arrive at a solution for the VPC, CILQR, and SQP algorithms. The inference time was shorter for VPC than MTUNet_1× (24.52 ms). Moreover, the CILQR had a computation speed that was faster than the ego vehicle control

Table 7 Dynamic system models and parameters for implementing the CILQR and SQP controllers

	Lateral CILQR-/SQP-based	Longitudinal CILQR/SQP
Dynamic model	Eq. (19)	Eq. (25)
Sampling time (dt)	0.05 s	0.1 s
Pred. horizon (N)	30	30
Ref. dist. (D_r)	–	11 m
Weighting matrixes	$\mathbf{Q}$ = diag (20, 1, 20, 1)	$\mathbf{Q}'$ = diag (20, 20, 1)
	$\mathbf{R} = [1]$	$\mathbf{R}' = [1]$

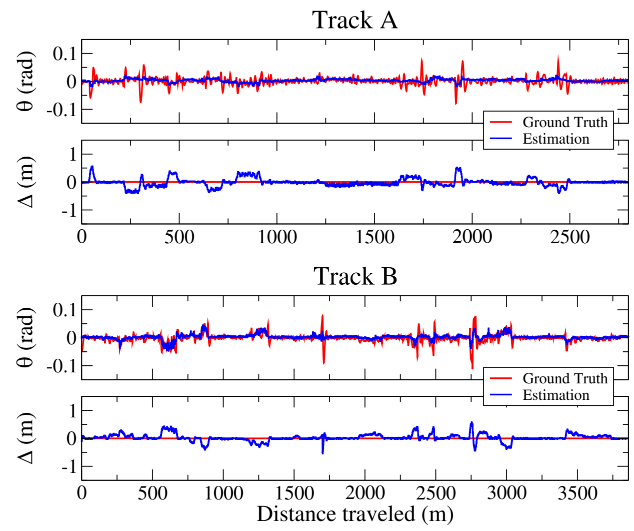

Fig. 6 Dynamic performance of lateral VPC-CILQR algorithm and MTUNet_1× model for an ego vehicle with heading θ and lateral offset Δ for lane-keeping maneuvers in the central lanes of Tracks A and B at 76 and 50 km/h, respectively. At the curviest section of Track A (near 1900 m), the maximal Δ value was 0.52 m; the ego car controlled by this model outperformed the ego car controlled by the Stanley controller (Fig. 3 in [28]), MTL-RL [Fig. 11(a) in [31]], or CILQR (Fig. 7) algorithms

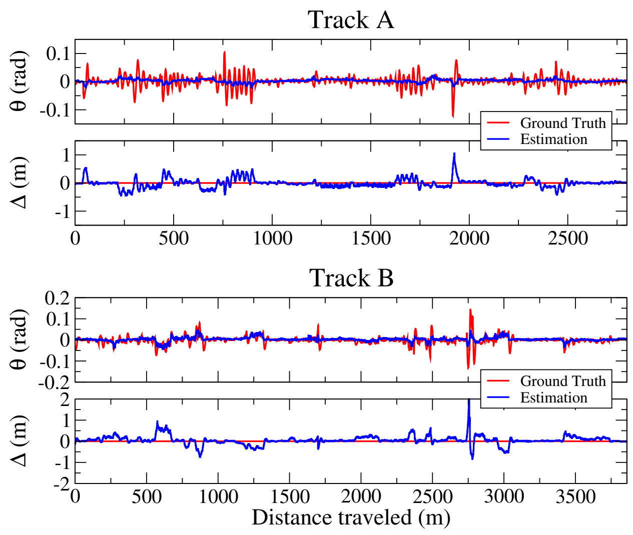

Fig. 8 Dynamic performance of the lateral VPC-SQP algorithm and MTUNet_1× model for lane-keeping maneuvers in the central lanes of Tracks A and B at the same speeds as those in Fig. 6. For the curviest sections of Tracks A and B (near 1900 and 2750 m, respectively), the performance of the VPC-SQP algorithm was inferior to those of the VPC-CILQR and CILQR algorithms (Figs. 6 and 7). These algorithms also outperformed the SQP algorithm (Fig. 9), indicating the effectiveness of the VPC algorithm

period (6.66 ms); the SQP solvers were slower. Specifically, the computation time per cycle for the lane-keeping and car-following tasks, respectively, for the SQP solvers were 16.7 and 21.5 times longer than those of the CILQR solvers. A discussion of the results for all the tested controllers are presented as follows.

In Figs. 6–9, all methods, including the MTUNet_1× model, could effectively guide the ego car to drive along

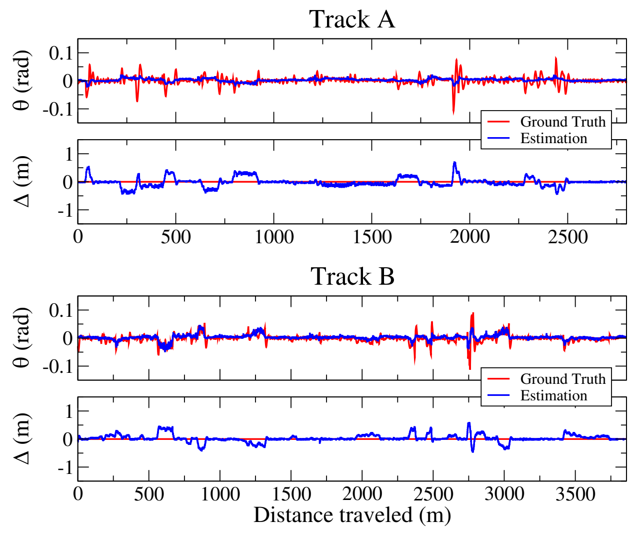

Fig. 7 Dynamic performance of the lateral CILQR algorithm and MTUNet_1× model for lane-keeping maneuvers in the central lanes of Tracks A and B at the same speeds as those in Fig. 6. At the curviest section of Track A (near 1900 m), the maximal Δ value was 0.71 m, which is 1.36 times larger than that of the ego car controlled by the VPC-CILQR algorithm

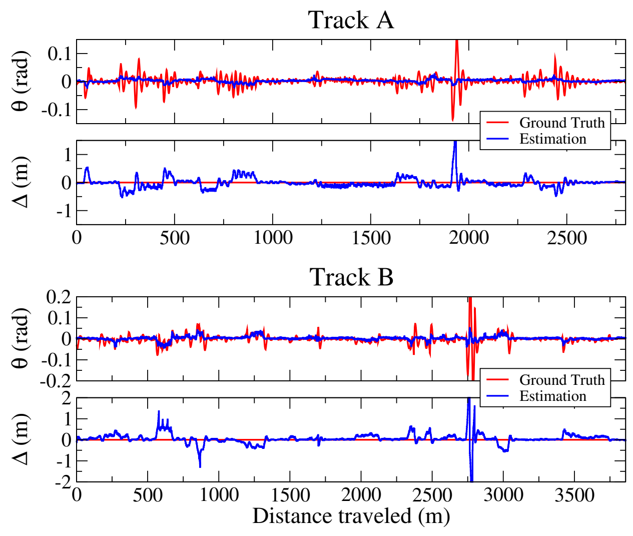

Fig. 9 Dynamic performance of the lateral SQP algorithm and MTUNet_1× model for lane-keeping maneuvers in the central lanes of Tracks A and B at the same speeds as those in Fig. 6. The model performance at the curviest section of Tracks A and B (near 1900 and 2750 m, respectively) was inferior to those of all other tested methods

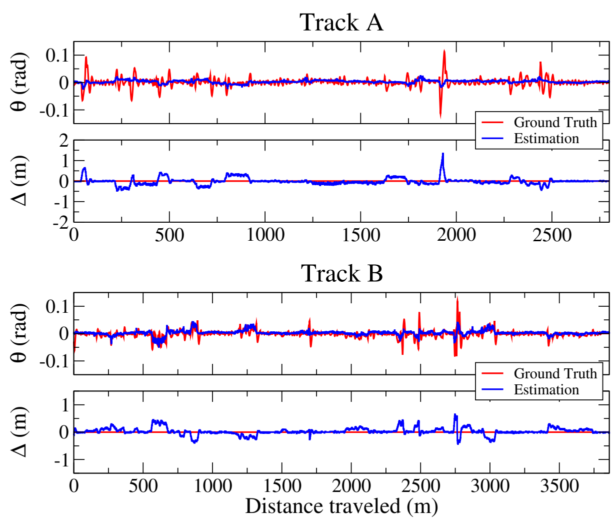

Fig. 10 Dynamic performance of the lateral VPC-CILQR algorithm and MTUNet_1× model for an ego vehicle with heading θ and lateral offset Δ for lane-keeping maneuvers in the central lanes of Tracks A and B at 80 and 60 km/h, respectively. At the curviest section of Track A, the maximal Δ value was 1.34 m

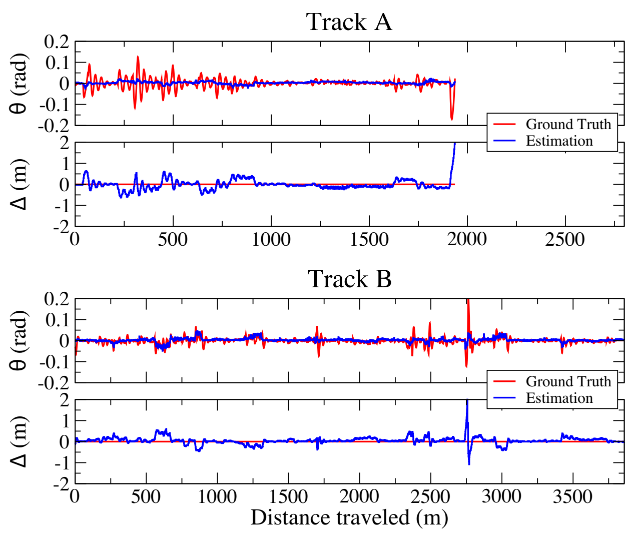

Fig. 12 Dynamic performance of the lateral VPC-SQP algorithm and MTUNet_1× model for lane-keeping maneuvers in the central lanes of Tracks A and B at the same speeds as those in Fig. 10

the lane center to complete one lap at cruise speeds of 76 and 50 km/h on Tracks A and B, respectively. The discrepancy in the θ between the MTUNet_1× estimation and the ground truth trajectory was attributable to curvy or shadowy road segments, which may induce vehicle jittering [31]. Nevertheless, the Δ values estimated from lane line segmentation were more robust in difficult scenarios than those obtained with the end-to-end method [61]. Therefore, these Δ values can be used by controllers to effectively correct θ errors and return the ego car to the road's center. The maximum Δ deviations from the ideal zero values on Track A (g-track-3 in [31]) were smaller when the ego car was controlled by the VPC-CILQR controller (shown in Fig. 6) than when it was controlled by the CILQR [21] (Fig. 7) or MTL-RL [31] algorithms. Note that the vehicle speed in the MTL-RL control framework on Track A for that study was 75 km/h, which is slower than that in this study. This finding indicates that for curvy roads, the VPC-CILQR algorithm better minimized Δ than did the other investigated algorithms. Due to the lower computation efficiency of the standard SQP solver [21], the

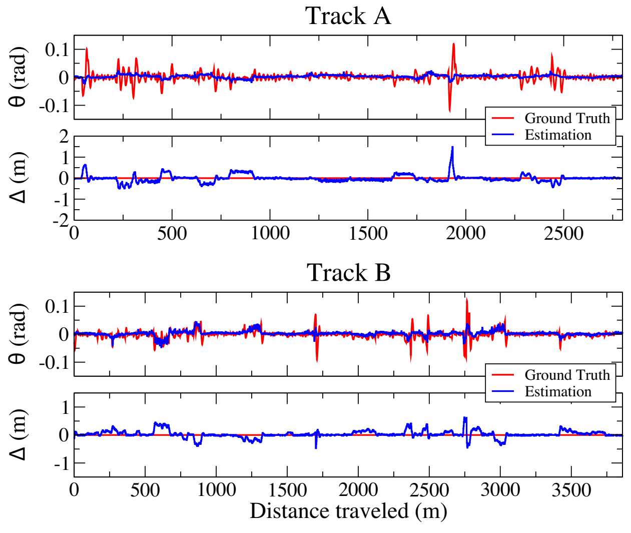

Fig. 11 Dynamic performance of the lateral CILQR algorithm and MTUNet_1× model for lane-keeping maneuvers in the central lanes of Tracks A and B at the same speeds as those in Fig. 10. At the curviest section of Track A, the maximal Δ value was 1.48 m

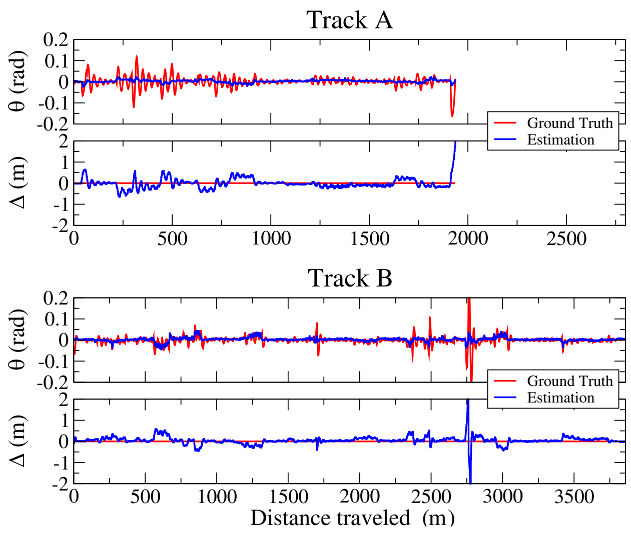

Fig. 13 Dynamic performance of the lateral SQP algorithm and MTUNet_1× model for lane-keeping maneuvers in the central lanes of Tracks A and B at the same speeds as those in Fig. 10

SQP-based controllers were less effective for maintaining the ego vehicle's stability than the CILQR-based controllers on the curviest sections of Tracks A and B (Figs. 8 and 9). Moreover, the VPC-SQP algorithm outperformed the SQP algorithm alone, further demonstrating the effectiveness of the VPC algorithm. In terms of MAE, the VPC-CILQR controller outperformed the other methods in terms of Δ-MAE on both tracks (data for 76 and 50 km/h in Table 8). However, θ-MAE was 0.0003 and 0.0005 rad higher on Tracks A and B, respectively, for the VPC-CILQR controller than the CILQR controller. This may have been because the optimality of CILQR solution is losing if the external VPC algorithm is applied to it. This problem could be solved by applying standard MPC methods with more general lane-keeping dynamics, such as the lateral control model presented in [52], which uses road curvature to describe vehicle states. This nonlinear MPC design is computationally expensive and may not meet the requirements for real-time autonomous driving.

In Figs. 10–13, the ego car was guided to drive along the central lane by the MTUNet_1× model at higher cruise speeds (80 and 60 km/h on Tracks A and B, respectively) than those in Figs. 6–9. For ego vehicles with the VPC-CILQR and CILQR controllers (Figs. 10 and 11), the maximum Δ deviations were approximately half of the lane width (2 m). In particular, the ego cars controlled by the SQP-based algorithms unintentionally left the ego-lane at the curviest section of Track A (Figs. 12 and 13). This was attributed to the slower reaction times of SQP-based algorithms (9.70 ms) than of CILQR-based algorithms (0.58 ms). Therefore, higher controller latency may not only result in ego car instability but also unsafe driving, particularly when the vehicle enters a curvy road at high speed.

The car-following maneuver in Fig. 14 was performed on a section of Track B. The ego vehicle was initially cruising at

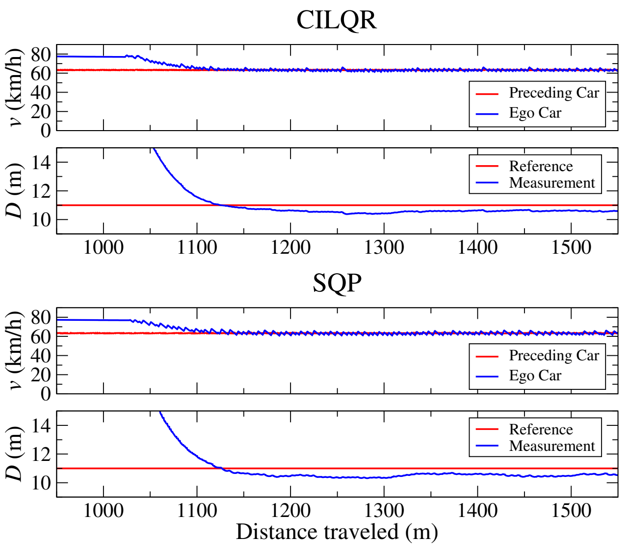

Fig. 14 Results for the longitudinal CILQR and SQP algorithms in car-following scenario after ego car travels 1075 m on Track B; v and D are speed and intervehicle distance, respectively

76 km/h and approached a slower preceding car with speed in the range of 63 to 64 km/h. For all ego vehicles with the CILQR or SQP controllers, the vehicle speed was regulated, the preceding vehicle was tracked, and the controller maintained a safe distance between the vehicles. However, the uncertainty in the optimal solution led to differences between the reference and response trajectories [18]. For the longitudinal CILQR and SQP controllers, respectively, v-MAE was 0.1971 and 0.2629 m/s, and D-MAE was 0.4201 and 0.4930 m (second row of Table 8). Hence, CILQR again outperformed SQP in this experiment. A supplementary video featuring the lane-keeping and car-following simulations can be found at https://youtu.be/Un-IJtCw83Q.

Table 8 Performance of the VPC-CILQR, CILQR, VPC-SQP, and SQP algorithms with MTUNet_1× in terms of the MAE for the tests in Figs. 6–14

Maneuver	Track	Speed (km/h)	VPC-CILQR		CILQR		VPC-SQP		SQP	
Lane-keeping			θ (rad)	Δ (m)	θ (rad)	Δ (m)	θ (rad)	Δ (m)	θ (rad)	Δ (m)
	A	76	0.0086	0.0980	0.0083	0.1058	0.0122	0.1071	0.0118	0.1187
	A	80	0.0099	0.1091	0.0098	0.1097	–	–	–	–
	B	50	0.0079	0.0748	0.0074	0.0775	0.0099	0.1286	0.0121	0.1470
	B	60	0.0078	0.0779	0.0079	0.0783	0.0099	0.1083	0.0112	0.1147
Car-following			v (m/s)	D (m)	v (m/s)	D (m)	v (m/s)	D (m)	v (m/s)	D (m)
	B[a]	–	–	–	0.1971	0.4201	–	–	0.2629	0.4930
Related trajectories			Figs. 6, 10		Figs. 7, 11, 14		Figs. 8, 12		Figs. 9, 13, 14	

[a]Computation from 1150 to 1550 m

Table 9 Average computation time of VPC, CILQR, and SQP algorithms

Task	VPC	CILQR	SQP
Lane-keeping	15.56 ms	0.58 ms	9.70 ms
Car-following	–	0.65 ms	14.01 ms

5 Conclusion

In this study, a vision-based self-driving system that uses a monocular camera and radars to collect sensing data is proposed; the system comprises an MTUNet network for environment perception and VPC and CILQR modules for motion planning. The proposed MTUNet model is an improvement on our previous model [28]; we have added a YOLOv4 detector and increased the network's efficiency by reducing the network input size for use with TORCS [28], CULane [56], and LLAMAS [57] data. The most efficient MTUNet model, namely MTUNet_1×, achieved an inference speed of 40.77 FPS for simultaneous lane line segmentation, ego vehicle pose estimation, and traffic object detection tasks. For vehicular automation, a lateral VPC-CILQR controller was designed that can plan vehicle motion based on the ego vehicle's heading, lateral offset, and road curvature as determined by MTUNet_1× and postprocessing methods. The longitudinal CILQR controller is activated when a slower preceding car is detected. The optimal jerk is then applied to regulate the ego vehicle's speed to prevent a collision. The MTUNet_1× and VPC-CILQR controller can collaborate for ego vehicle operation on challenging tracks in TORCS; this algorithm outperforms methods based on the CILQR [21] or MTL-RL [31] algorithms for the same path-tracking task on the same large-curvature roads. Moreover, the self-driving vehicle with long-latency SQP-based controllers tended to leave the lane on some curvy routes. By contrast, the short-latency CILQR-based controllers could drive stably and safely in the same scenarios. In conclusion, the experiments demonstrated the applicability and feasibility of the proposed system, which comprises perception, planning, and control algorithms. It is suitable for real-time autonomous vehicle control and does not require HD maps. A future study can apply the proposed autonomous driving system to a real vehicle operating on actual roads.

Funding The author did not receive support from any organization for the submitted work.

Code Availibility Statement A sample of TORCS dataset containing 500 images with corresponding labels can be found at https://drive.google.com/file/d/1iAAH5dH2YiAx_LBr4BiOkiwl1HqKL9NU/view?usp=sharing publicly.

Declarations

Conflicts of interest The author declares that there is no conflict of interest regarding the publication of this article.

References

1. Grigorescu S, Trasnea B, Cocias T, Macesanu G (2020) A survey of deep learning techniques for autonomous driving. J Field Robot 37(3):62–386
2. Yurtsever E, Lambert J, Carballo A, Takeda K (2020) A Survey of Autonomous Driving: Common Practices and Emerging Technologies. IEEE Access 8:58443–58469
3. Tampuu A, Matiisen T, Semikin M, Fishman D, Muhammad N (2022) A Survey of End-to-End Driving: Architectures and Training Methods. IEEE Trans Neural Netw Learn Syst 33(4):1364–1384
4. Li Y, Ibanez-Guzman J (2020) Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems. IEEE Signal Process Mag 37(4):50–61
5. (2020) Active Driving Assistance Systems: Test Results and Design Recommendations. Consumer Reports
6. Teichmann M, Weber M, Zöllner M, Cipolla R, Urtasun R (2018) MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving. In: IEEE Intelligent Vehicles Symposium (IV), pp. 1013-1020
7. Pizzati F, García F (2019) Enhanced free space detection in multiple lanes based on single CNN with scene identification. In: IEEE Intelligent Vehicles Symposium (IV), pp. 2536-2541
8. Qian Y, Dolan JM, Yang M (2020) DLT-Net: Joint Detection of Drivable Areas, Lane Lines, and Traffic Objects. IEEE Trans Intell Transp Syst 21(11):4670–4679
9. Lai C-Y, Wu B-X, Shivanna VM, Guo J-I (2021) MTSAN: Multi-Task Semantic Attention Network for ADAS Applications. IEEE Access 9:50700–50714
10. Wu D, Liao M, Zhang W, Wang X, Bai X, Cheng W, Liu W (2022) YOLOP: You Only Look Once for Panoptic Driving Perception. Mach Intell Res 19:550–562
11. Artificial Intelligence & Autopilot. https://www.tesla.com/AI. Accessed 21 May 2023
12. Paden B, Čáp M, Yong SZ, Yershov D, Frazzoli E (2016) A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles. IEEE Trans Intell Veh 1(1):33–55
13. Lim W, Lee S, Sunwoo M, Jo K (2021) Hybrid Trajectory Planning for Autonomous Driving in On-Road Dynamic Scenarios. IEEE Trans Intell Transp Syst 22(1):341–355
14. Gutjahr B, Gröll L, Werling M (2017) Lateral Vehicle Trajectory Optimization Using Constrained Linear Time-Varying MPC. IEEE Trans Intell Transp Syst 18(6):1586–1595
15. Turri V, Carvalho A, Tseng HE, Johansson KH, Borrelli F (2013) Linear model predictive control for lane keeping and obstacle avoidance on low curvature roads. In: 16th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp. 378-383
16. Katriniok A, Maschuw JP, Christen F, Eckstein L, Abel D (2013) Optimal vehicle dynamics control for combined longitudinal and lateral autonomous vehicle guidance. In: European Control Conference (ECC), pp. 974-979
17. Li SE, Jia Z, Li K, Cheng B (2015) Fast Online Computation of a Model Predictive Controller and Its Application to Fuel Economy-

Oriented Adaptive Cruise Control. IEEE Trans Intell Transp Syst 16(3):1199–1209
18. Lim W, Lee S, Yang J, Sunwoo M, Na Y, Jo K (2022) Automatic Weight Determination in Model Predictive Control for Personalized Car-Following Control. IEEE Access 10:19812–19824
19. Mattingley J, Boyd S (2012) CVXGEN: A code generator for embedded convex optimization. Optim Eng 13(1):1–27
20. Chen J, Zhan W, Tomizuka M (2017) Constrained iterative LQR for on-road autonomous driving motion planning. In: IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), pp. 1-7
21. Chen J, Zhan W, Tomizuka M (2019) Autonomous Driving Motion Planning With Constrained Iterative LQR. IEEE Trans Intell Veh 4(2):244–254
22. Jacobson DH, Mayne DQ (1970) Differential Dynamic Programming. Elsevier
23. Pan Y, Lin Q, Shah H, Dolan J M (2020) Safe Planning for Self-Driving Via Adaptive Constrained ILQR. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2377-2383
24. Ma J, Cheng Z, Zhang X, Lin Z, Lewis FL, Lee TH (2023) Local Learning Enabled Iterative Linear Quadratic Regulator for Constrained Trajectory Planning. IEEE Trans Neural Netw Learn Syst 34(9):5354–5365
25. Werling M, Ziegler J, Kammel S, Thrun S (2010) Optimal trajectory generation for dynamic street scenarios in a Frenét frame. In:IEEE International Conference on Robotics and Automation (ICRA), pp. 987-993
26. Lee K, Jeon S, Kim H, Kum D (2019) Optimal Path Tracking Control of Autonomous Vehicle: Adaptive Full-State Linear Quadratic Gaussian (LQG) Control. IEEE Access 7:109120–109133
27. Lee D-H, Chen K-L, Liou K-H, Liu C-L, Liu J-L (2021) Deep learning and control algorithms of direct perception for autonomous driving. Appl Intell 51:237–247
28. Lee D-H, Liu C-L (2021) Multi-task UNet architecture for end-to-end autonomous driving. arXiv:2112.08967
29. Lee D-H, Liu C-L (2023) End-to-end deep learning of lane detection and path prediction for real-time autonomous driving. SIViP 17:199–205
30. Thrun S et al (2006) Stanley: The robot that won the DARPA grand challenge. J Field Robot 23(9):661–692
31. Li D, Zhao D, Zhang Q, Chen Y (2019) Reinforcement learning and deep learning based lateral control for autonomous driving. IEEE Comput Intell Mag 14(2):83–98
32. Liu J, Yang Z, Huang Z, Li W, Dang S and Li H (2021) Simulation Performance Evaluation of Pure Pursuit, Stanley, LQR, MPC Controller for Autonomous Vehicles. In: IEEE International Conference on Real-time Computing and Robotics (RCAR), pp. 1444-1449
33. Lu P, Cui C, Xu S, Peng H, Wang F (2021) SUPER: A Novel Lane Detection System. IEEE Trans Intell Veh 6(3):583–593
34. Wang C-Y, Bochkovskiy A, Mark Liao H-Y (2021) Scaled-YOLOv4: Scaling Cross Stage Partial Network. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 13029-13038
35. Georg J M, Feiler J, Hoffmann S, Diermeyer F (2020) Sensor and Actuator Latency during Teleoperation of Automated Vehicles. In: IEEE Intelligent Vehicles Symposium (IV), pp. 760-766
36. Betz J et al (2023) TUM autonomous motorsport: An autonomous racing software for the Indy Autonomous Challenge. J Field Robot 40:783–809
37. Ibtehaz N, Rahman MS (2020) MultiResUNet: Rethinking the U-Net architecture for multimodal biomedical image segmentation. Neural Netw 121:74–87
38. Simonyan K, Zisserman A (2015) Very Deep Convolutional Networks for Large-Scale Image Recognition. In: International Conference on Learning Representations (ICLR), pp. 1-14
39. Ronneberger O, Fischer P, Brox T (2015) U-Net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI). Springer, pp. 234-241
40. Bruls T, Maddern W, Morye AA, Newman P (2018) Mark yourself: Road marking segmentation via weakly supervised annotations from multimodal data. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 1863-1870
41. Xie S, Tu Z (2015) Holistically-nested edge detection. In IEEE International Conference on Computer Vision (ICCV), pp. 1395-1403
42. Zou Q, Jiang H, Dai Q, Yue Y, Chen L, Wang Q (2020) Robust lane detection from continuous driving scenes using deep neural networks. IEEE Trans Veh Technol 69(1):41–54
43. Cui H, Radosavljevic V, Chou F-C, Lin T-H, Nguyen T, Huang T-K, Schneider J, Djuric N (2019) Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In: International Conference on Robotics and Automation (ICRA), pp. 2090-2096
44. Zheng Z, Wang P, Liu W, Li J, Ye R, Ren D (2020) Distance-IoU Loss: Faster and better learning for bounding box regression. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pp. 12993-13000
45. Choi JI, Tian Q (2022) Adversarial Attack and Defense of YOLO Detectors in Autonomous Driving Scenarios. In: IEEE Intelligent Vehicles Symposium (IV), pp. 1011-1017
46. Tassa Y, Erez T, Todorov E (2012) Synthesis and stabilization of complex behaviors through online trajectory optimization. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4906-4913
47. Tassa Y, Mansard N, Todorov E (2014) Control-limited differential dynamic programming. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 1168-1175
48. Plancher B, Manchester Z, Kuindersma S (2017) Constrained unscented dynamic programming. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5674-5680
49. Samak C V, Samak T V, and Kandhasamy S (2021) Control Strategies for Autonomous Vehicles. In: Autonomous Driving and Advanced Driver-Assistance Systems, CRC Press
50. Qiu W, Ting Q, Shuyou Y, Hongyan G, and Hong C (2015) Autonomous vehicle longitudinal following control based on model predictive control. In: IEEE 34th Chinese Control Conference (CCC), pp. 8126-8131
51. Zhang X, Zhu X (2019) Autonomous path tracking control of intelligent electric vehicles based on lane detection and optimal preview method. Expert Syst Appl 121:38–48
52. Xu S, Peng H, Lu P, Zhu M, Tang Y (2020) Design and Experiments of Safeguard Protected Preview Lane Keeping Control for Autonomous Vehicles. IEEE Access 8:29944–29953
53. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu C, Berg A (2016) SSD: Single Shot MultiBox Detector. In: European Conference on Computer Vision (ECCV). pp. 21-37
54. Redmon J, Farhadi A (2017) YOLO9000: Better, Faster, Stronger. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6517-6525
55. Lee S et al (2017) VPGNet: Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition. In: IEEE International Conference on Computer Vision (ICCV), pp. 1965-1973
56. Pan X, Shi J, Luo P, Wang X, Tang X (2018) Spatial as deep: spatial CNN for traffic scene understanding. In: The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), pp. 7276-7283

57. Behrendt K, Soussan R (2019) Unsupervised Labeled Lane Markers Using Maps. In: IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 832-839
58. MacQueen J et al (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, pp. 281-297
59. Bernhard W et al (2014) TORCS: The open racing car simulator. http://www.torcs.org
60. Fitzpatrick K (1994) Horizontal Curve Design: An Exercise in Comfort and Appearance. Transp Res Rec 1445:47–53
61. Chen C, Seff A, Kornhauser A, Xiao J (2015) DeepDriving: Learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2722-2730
62. Standley T, Zamir A R, Chen D, Guibas L, Malik J, Savarese S (2020) Which tasks should be learned together in multi-task learning? In: International Conference on Machine Learning (ICML), pp. 9120-9132
63. Kokkinos I (2017) UberNet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5454-5463
64. Bianco S, Cadene R, Celona L, Napoletano P (2018) Benchmark Analysis of Representative Deep Neural Network Architectures. IEEE Access 6:64270–64277
65. Houska B, Ferreau HJ, Diehl M (2011) ACADO toolkit-An open source framework for automatic control and dynamic optimization. Optim Control Appl Methods 32(3):298–312

Der-Hau Lee received his Ph.D. degree in Physics from the Department of Electrophysics, National Yang Ming Chiao Tung University, Taiwan, in 2018. His research interests include machine intelligence and autonomous driving.