IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

# D2MFusion: An End-to-End Differentiable Trajectory Optimizer for Safe Reactive Navigation

Xiangyu Zhou, Shenghong Zhang, and Xiao Li

***Abstract*—Data-driven methods provide effective solutions for robot trajectory generation in dynamic environments. Many physical constraints exist in the real world, and understanding these constraints to generate feasible trajectories for kinematics or dynamics is highly demanding regarding the data quantity. Due to the black box, it is also challenging to ensure the safety of the trajectories planned by data-driven models. In this paper, we propose an end-to-end model (*D2MFusion*) that fuses *data-driven components* and a *model-based optimizer*. *D2MFusion* uses a differentiable optimization layer (dLQR) that forms a backpropagation loop with a perception network. With the input BEV image, the perception network outputs the environmental feature vector to adjust the optimizer parameters to adapt to the dynamic environment. We train this fusion planner to imitate expert trajectories on a real self-driving dataset and demonstrate the planner's explainability, data efficiency, and safe reactivity through closed-loop simulations. We also conduct experiments on a Unitree Go2 in three different scenarios to demonstrate the ability of our method to navigate in dynamic environments.**

***Index Terms*—Safe Reactive Navigation, End-to-End Differentiable Trajectory Optimizer, Planning and Learning.**

## I. Introduction

In recent years, robots have been increasingly deployed in complex open environments such as shopping malls, museums, streets, and medical care centers [1]. These environments involve dynamic risks and uncertainties [2], posing significant challenges for safe reactive navigation. To address these challenges, trajectory planners are expected to accurately detect and respond to environmental changes. Data-driven approaches leverage large-scale multi-modal data to sense and predict dynamics, enabling more adaptive planning. However, they often require extensive datasets to capture rich real-world constraints, leading to high model complexity. Moreover, their black-box nature makes it difficult to generalize to rare or unseen situations, raising safety concerns. In this work, we propose an end-to-end trainable planner that improves explainability, data efficiency, and safe reactivity in dynamic environments.

Traditional model-based planning methods, such as model predictive control, explicitly incorporate kinematic, dynamical, and boundary constraints into the planning process, ensuring physically feasible trajectories while reducing data requirements and enhancing safety and reliability. However, the parameters of model-based planners are often difficult to tune and adapt, limiting their flexibility in dynamic environments. The combination of the efficiency and safety of model-based methods with the intelligence of data-driven methods has thus become a practical and active research direction. Most of the existing work integrates neural networks into model-based planners in specific components, such as fitting cost functions [3], [4], approximating nonlinear dynamics [5], or generating maps for planning [6]. In contrast, fewer studies have explored the profound integration of the two methods. This paper investigates a fusion method that combines a neural network with a model-based optimizer.

In this paper, we propose a differentiable architecture as shown in Fig. 1. BEV images, a compact representation bridging perception and planning [7], serve as inputs to the perception network. The perception module extracts the feature vector encoding environmental information, which is optimized and transformed before being passed as parameters to the downstream differentiable planner. The planner (dLQR) then generates trajectories. For this component, we adopt OptNet [8], an implicit neural network layer for end-to-end learning. To summarize our contributions, we

- introduce the *D2MFusion* - a differentiable planning architecture that achieves safe reactive navigation and high explainability by adjusting the planning strategy through a feature vector reflecting the reactive attitude towards itself and the surrounding dynamic objects;
- introduce a fusing method for neural networks and differentiable optimizers that can realize learning end-to-end and data efficiency by incorporating the dLQR in the backpropagation;
- demonstrate the planner's **explainability**, **data efficiency**, and **safe reactivity** on the nuPlan self-driving dataset and validate the effectiveness of the method in dynamic navigation experiments on a real robot.

In this work, we focus on applying the neural planner to safe reactive navigation in ***generic dynamic environments***, such as real-world robotics and autonomous driving.

## II. Related Work

**Data-driven planner.** Data-driven planners have recently advanced robot navigation and autonomous driving. [9] employs a CNN to learn safe driving behaviors, directly outputting steering actions. [10] introduces an end-to-end multi-modal network for semantic segmentation and control. Similarly, [11] shows that incorporating more low-quality data can

Manuscript received: June 19, 2025; Accepted: September 2, 2025.

This paper was recommended for publication by Editor Olivier Stasse upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by the National Key R&D Program of China (Grant No. 2024YFB4707400) and by NSFC grant #52405029. *(Corresponding author: Xiao Li.)*

The authors are with the Institute of Robotics, School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai, China (email: xiangyuzhou@sjtu.edu.cn; zsh000@sjtu.edu.cn; sjtu_lixiao@sjtu.edu.cn).

Digital Object Identifier (DOI): see top of this page.

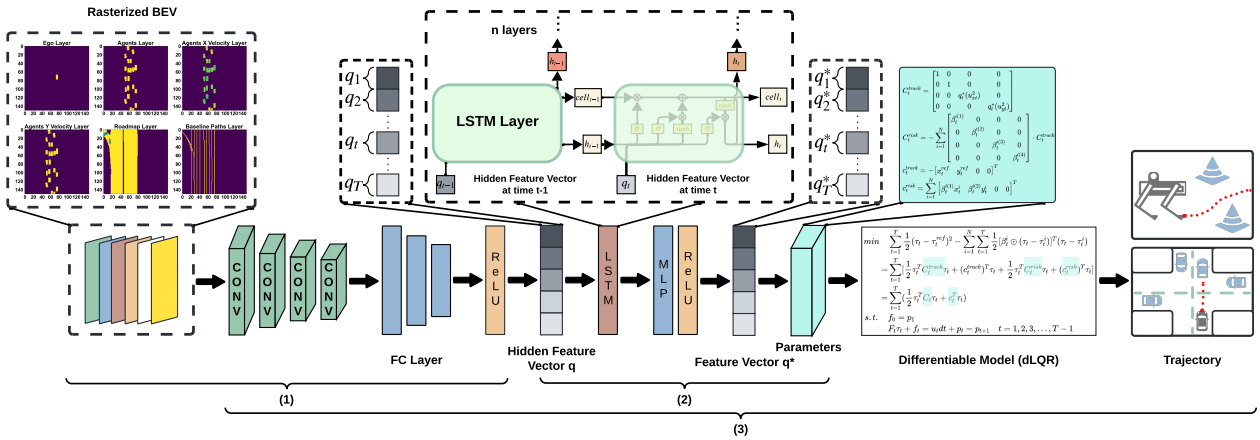


Fig. 1. **End-to-end data-driven and dLQR fusing planner.** The architecture contains three components: (1) a CNN-based perception network that extracts environmental information from BEV images to produce a hidden feature vector; (2) a module that optimizes and transforms this vector into parameters for the downstream differentiable planner; and (3) a differentiable planner with its parameters controlled by the feature vector to form a gradient loop.

improve training. For prediction and planning, [12] proposes a model that jointly predicts the ego and other agents' trajectories, while [13] presents TOFG, a unified representation of HD maps and trajectories with time, enhancing vehicle-lane and vehicle-vehicle interaction modeling. Data-driven planners also extend to multi-robot planning. D2CoPlan [14], a distributed differentiable planner, demonstrates scalability in UAV coverage tasks. [15] improves driving success rates by aggregating critical states with a replay buffer. In driving risk perception, [16] learns a risk map with CNNs, applies IRL for policy learning, and samples risk-avoidance trajectories. However, these methods remain black-box and lack constraints for safety. *Our work enhances safety in dynamic environments and achieves data efficiency through the integration of dLQR.*

**Combination of data-driven and model-based methods.** [17] integrates a guided map generator with a differentiable $A^*$ for end-to-end training, but its CNN-based generator is less effective for large maps. [6] improves scalability by employing a ViT in place of CNNs. While these methods can compute optimal global paths, they are limited in dynamic environments. In MPC, [5] approximates nonlinear dynamics using a neural network, and [18] learns control strategies with neural networks; in both cases, the network substitutes only part of the optimizer without deeper integration with model-based components. In multi-robot perception, [19] employs an RNN to predict surrounding trajectories, which are then processed by a distributed controller, again without coupling between network and controller. [20] introduces a feedback synthesizer with differentiable rasterization, embedding kinematics as a differentiable layer. In reinforcement learning, [21] proposes BRSL, which combines data-driven reachability analysis and differentiable collision detection to adjust policy behavior, but it relies on specific assumptions and approximations. DAN [22] provides a structured, composable end-to-end differentiable architecture, though the information flow among modules is not fully interpretable. DiffStack [23] integrates prediction, planning, and control by incorporating predicted trajectories into a differentiable planning–control cost, but its explainability remains limited. Our previous work [24] proposes a multi-abstractive neural controller, yet the semantics of q-states remain implicit, and the explainability is affected by the number of q-states. [25] presents a differentiable framework with a learnable cost for joint prediction and planning, but the training process is complex and not explainable. Most of these approaches also lack real-world validation. iplanner [26] achieves real-time path planning from depth data by exploiting implicit costs learned through imperative learning, which differs from our approach that adapts planner parameters to dynamic environmental features. *Our work introduces a feature vector-based fusion framework with an emphasis on explainability.*

## III. BACKGROUND

### A. LQR Considering the Risk of Surrounding Agents

The LQR problem that considers the risk of multiple agents around our controlled robot (ego) is derived as below.

$$\begin{aligned} \min \quad & \sum_{t=1}^{T} \frac{(\tau_t - \tau_t^{ref})^2}{2} - \sum_{i=1}^{N}\sum_{t=1}^{T} \frac{[\beta_t^i \odot (\tau_t - \tau_t^i)]^T (\tau_t - \tau_t^i)}{2} \\ \text{s.t.} \quad & f_0 = p_1 \\ & F_t\tau_t + f_t = u_t dt + p_t = p_{t+1} \quad t = 1, 2, 3, ..., T-1 \end{aligned} \tag{1}$$

The cost function consists of the tracking error term and the risk term that presents the negative distances between the agents and the ego. The optimization goal is simultaneously satisfying the need to minimize the tracking error and keep a safe distance from the risky agents. The vectors used in this paper are, by default, column vectors. In the cost function, $T$ defines the number of time points for planning, $\tau_t \in \mathbb{R}^4$ is the stack of 2D state $p_t = [x_t \quad y_t]^T \in \mathbb{R}^2$ and 2D control $u_t = [u_{xt} \quad u_{yt}]^T \in \mathbb{R}^2$ of the ego at the moment $t$, $x_t$ refers to the state of the ego in the current forward direction $x$, $y_t$ refers to the state of the ego in the current sliding sideways direction $y$, $u_{xt}$ refers to the speed control in the direction $x$, and $u_{yt}$ refers to the speed control in the direction $y$. $\tau_t^{ref} \in \mathbb{R}^4$ gives the reference state and control of the ego at the moment $t$, $N$ is the number of the risk agents around the ego, and $\tau_t^i \in \mathbb{R}^4$

is the stack of the state and the control of the $i$ agent at the moment $t$. $\beta_t^i = [\beta_t^{i(1)} \quad \beta_t^{i(2)} \quad \beta_t^{i(3)} \quad \beta_t^{i(4)}]^T \in \mathbb{R}^4$ is defined as the weight vector to be attached to the negative distance of the $i$ agent at moment $t$. The $\odot$ is the Hadamard product. In the constraints, $p_1$ is the initial state of the ego, and $dt$ is the control step. The kinematic equation is a point motion model of uniform motion where $u_t$ controls $p_t$ and is organized into a linear form [27], where $F_t$ is the primary coefficient and $f_t$ is the constant term.

The tracking error term of the cost function is developed and organized as follows, and the following notations are defined.

$$\sum_{t=1}^{T} \frac{(\tau_t - \tau_t^{ref})^2}{2} = \sum_{t=1}^{T} [\frac{\tau_t^T C_t^{track} \tau_t}{2} - (\tau_t^{ref})^T \tau_t + \frac{(\tau_t^{ref})^2}{2}] \tag{2}$$

$$C_t^{track} = \mathrm{I}_4, \quad c_t^{track} = -\tau_t^{ref} \tag{3}$$

The risk term of the cost function is developed and organized as follows:

$$\begin{aligned} &-\sum_{i=1}^{N}\sum_{t=1}^{T} \frac{1}{2}[\beta_t^i \odot (\tau_t - \tau_t^i)]^T (\tau_t - \tau_t^i) \\ =&\sum_{t=1}^{T}[-\frac{1}{2}\tau_t^T \sum_{i=1}^{N}(\mathrm{diag}(\beta_t^{i(1)}, \beta_t^{i(2)}, \beta_t^{i(3)}, \beta_t^{i(4)}) \cdot C_t^{track})\tau_t \\ &+\sum_{i=1}^{N}(\beta_t^i \odot \tau_t^i)^T \tau_t - \frac{1}{2}\sum_{i=1}^{N}(\beta_t^i \odot \tau_t^i)^T \tau_t^i] \end{aligned} \tag{4}$$

$C_t^{track}$ is an identity matrix so that the above equation can be multiplied by $C_t^{track}$. $(\tau_t^{ref})^2$ and $(\beta_t^i \odot \tau_t^i)^T \tau_t^i$ are constant terms that do not work in minimizing the cost function, so they are eliminated. Define the notations as follows.

$$C_t^{risk\_i} = \mathrm{diag}(\beta_t^{i(1)}, \beta_t^{i(2)}, \beta_t^{i(3)}, \beta_t^{i(4)}) \cdot C_t^{track} \tag{5}$$

$$C_t^{risk} = -\sum_{i=1}^{N} C_t^{risk\_i}, \quad c_t^{risk} = \sum_{i=1}^{N}(\beta_t^i \odot \tau_t^i) \tag{6}$$

The LQR problem can be organized into the following simple form based on the derivation and the defined notations.

$$\begin{aligned} \min \quad & \sum_{t=1}^{T}[\frac{1}{2}\tau_t^T (C_t^{track} + C_t^{risk})\tau_t + (c_t^{track} + c_t^{risk})^T \tau_t] \\ =&\sum_{t=1}^{T}(\frac{1}{2}\tau_t^T C_t \tau_t + c_t^T \tau_t) \\ \text{s.t.} \quad & f_0 = p_1 \\ & F_t\tau_t + f_t = u_t dt + p_t = p_{t+1} \quad t = 1, 2, 3, ..., T-1 \end{aligned} \tag{7}$$

where $C_t = C_t^{track} + C_t^{risk} \in \mathbb{R}^4 \times \mathbb{R}^4$ is the quadratic coefficient integrating the tracking error term and the risk term, and $c_t = c_t^{track} + c_t^{risk} \in \mathbb{R}^4$ is the linear coefficient integrating these two terms.

### B. Differentiable Linear Quadratic Regulator (dLQR)

The previous subsection introduced the LQR optimization problem considering the risk of agents, defining the set of LQR parameters as $\mathcal{W}$, denoted as $\{C_t,\ c_t,\ F_t,\ f_t\}$. The authors of [8] and [28] presented the LQR as a layer of a neural network, giving the $\partial loss/\partial \mathcal{W}$. The derivation procedure of the gradient information of $loss$ with respect to each parameter $\mathcal{W}$ in dLQR can be viewed in [8] and [28].

## IV. End-to-End Data-Driven and DLQR Fusing Planner

In this section, we introduce the overall architecture and three components of the end-to-end fusing planner using autonomous driving scenarios as an example. Actually, the planner can also be applied for safe reactive navigation of robots in dynamic environments, as subsequently shown in real-world experiments. The overall architecture is shown in Fig. 1: (1) a CNN-based perception network that extracts the environmental information in the BEV image to generate a hidden feature vector; (2) a component that optimizes the hidden feature vector and transforms it into the parameters of a downstream differentiable planner; and (3) a differentiable planner with its parameters controlled by the feature vector to form a gradient loop.

### A. Feature Vector Extractor

The feature extractor generates a hidden feature vector that reflects the preliminary attitude of trajectory planning for the current frame based on the rich environmental information in the BEV image [7]. The BEV consists of six channels, as shown in Fig. 1: ego map, agents map, velocity map of agents in the forward direction $x$ of ego, velocity map of agents in the sideways direction $y$ of ego, road map, and lane map. The velocity components are calculated as follows.

$$^{ego}V_x = {}^{W}_{ego}T^{-1} \cdot {}^{W}V_x, \quad ^{ego}V_y = {}^{W}_{ego}T^{-1} \cdot {}^{W}V_y \tag{8}$$

where $W$ is the world coordinate system. Given the above BEV maps (denoted as $\mathcal{X}$) as the input, the hidden feature vector is obtained by the following convolutional neural network structure.

$$\begin{aligned} \boldsymbol{p} &= \texttt{ConvEncoder}(\mathcal{X}) \\ \boldsymbol{q} &= \texttt{ReLU}\big(\texttt{MultiLinear}(\boldsymbol{p})\big) \end{aligned} \tag{9}$$

In the above equation, the $\texttt{ConvEncoder}(\cdot)$ is a pre-trained resnet50 [29] that encodes the rasterized BEV image and outputs a high-dimensional vector through the fully-connected layer inside the resnet50. Passing this high-dimensional vector through multiple linear transformation layers $\texttt{MultiLinear}(\cdot)$ contributes to regressing the physical meaning we need. Finally, the non-negativity of the vector is guaranteed by $\texttt{ReLU}(\cdot)$. This component has extracted a hidden feature vector $\boldsymbol{q}$, which contains the current environmental information and subsequently needs to be optimized.

### B. Embedding Feature Vector Optimization

The LQR in III-A is with different planner parameters at the moment $1 \sim T$, so $\boldsymbol{q}$ needs to be decomposed into $T$ $\boldsymbol{q_t}$ vectors with the same dimension, i.e., $\boldsymbol{q}$ is a sequence consisting of $\boldsymbol{q_t}$ component from the moment 1 to the moment $T$ (as shown in Fig. 1). However, the hidden feature vector $\boldsymbol{q}$

cannot be directly used for the parameters of the dLQR planner because the planner parameters are related to each other over different time windows, but $\boldsymbol{q_t}$s are not. If used directly, it would cause the strategy in the current time window to be independent of the previous and later time windows, which may lead to acceleration fluctuations (discussed in Section V). We apply LSTM to optimize the hidden feature vector. Define the feature vector processed by LSTM as $\boldsymbol{q^*}$. Similarly, $\boldsymbol{q^*}$ is decomposed into $T$ $\boldsymbol{q_t^*}$ components at the moment $1\sim T$, and can be expressed as $\boldsymbol{q^*} = [(\boldsymbol{q_1^*})^T \cdots (\boldsymbol{q_t^*})^T \cdots (\boldsymbol{q_T^*})^T]^T$. The optimization process is as follows.

$$\boldsymbol{q^*} = \texttt{ReLU}\big(\texttt{MLP}(\texttt{LSTM}(\boldsymbol{q}))\big) \tag{10}$$

In III-A, we focus on $N$ agents. For the $C_t^{risk\_i}$ matrix in Eq. (5) of each $i$ agent, the corresponding $\beta_t^i \in \mathbb{R}^4$ contains four weights ($[\beta_t^{i(1)} \quad \beta_t^{i(2)} \quad \beta_t^{i(3)} \quad \beta_t^{i(4)}]^T$). Specifically, in Eq. (4), $\beta_t^{i(1)}$, $\beta_t^{i(2)}$, $\beta_t^{i(3)}$, $\beta_t^{i(4)}$ are the weights of the squared terms ($x_t^2$, $y_t^2$, $u_{xt}^2$, $u_{yt}^2$ ) of $\tau_t$ in the risk term, and similarly these four components multiplied by $\tau_t^i$ are the weights of the primary terms ($x_t$, $y_t$, $u_{xt}$, $u_{yt}$) of $\tau_t$ in the risk term. In addition, the reference trajectory uses lane centerline coordinates during training and lacks a reference for control. In fact, in the real world, we usually can only provide the reference trajectory coordinates without control as well. This leads to a more critical effect of the two control quadratic penalty terms $u_{xt}^2$ and $u_{yt}^2$ without $u_{xt}^{ref}u_{xt}$ and $u_{yt}^{ref}u_{yt}$ to weaken their effect in the tracking error term. The weights of these two terms can adjust the magnitude of the control quantities in the optimization process and thus affect the states, so $u_{xt}^2$ and $u_{yt}^2$ also require corresponding weights ($q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$) in $C_t^{track}$. From the above, the dLQR parameters at a certain moment $t$ require a $4 \times N + 2$ dimensional $\boldsymbol{q_t^*}$ vector [$q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$, $\beta_t^{1(1)}$, $\beta_t^{1(2)}$, $\beta_t^{1(3)}$, $\beta_t^{1(4)}$, $\cdots$, $\beta_t^{i(1)}$, $\beta_t^{i(2)}$, $\beta_t^{i(3)}$, $\beta_t^{i(4)}$, $\cdots$]$^T$ which contains the weights of the two control quadratic penalty terms described earlier and the $\beta_t^i$s of the $N$ agents, thus the vector dimension of the $\texttt{LSTM}(\cdot)$ output needs to be $T \times (4 \times N + 2)$. The $\texttt{LSTM}(\cdot)$ output is then followed by a linear transformation layer $\texttt{MLP}(\cdot)$ and a $\texttt{ReLU}(\cdot)$ to obtain the optimized feature vector $\boldsymbol{q^*}$.

The parameters in dLQR are presented in Eq. (7), which contain $C_t$, $c_t$, $F_t$, and $f_t$. Based on our analysis of the components of $\boldsymbol{q_t^*}$ above, Eq. (3), and Eq. (6), the transformation of $\boldsymbol{q_t^*}$ to each dLQR parameter is shown in Eq. (11).

$$\begin{aligned}
C_t^{track} &= \mathrm{diag}(1, 1, q_t^*(u_{xt}^2), q_t^*(u_{yt}^2)) \\
C_t^{risk} &= -\sum_{i=1}^{N} \mathrm{diag}(\beta_t^{i(1)}, \beta_t^{i(2)}, \beta_t^{i(3)}, \beta_t^{i(4)}) \cdot C_t^{track} \\
c_t^{track} &= -\begin{bmatrix} x_t^{ref} & y_t^{ref} & 0 & 0 \end{bmatrix}^T \\
c_t^{risk} &= \sum_{i=1}^{N} \begin{bmatrix} \beta_t^{i(1)} x_t^i & \beta_t^{i(2)} y_t^i & 0 & 0 \end{bmatrix}^T \\
C_t &= C_t^{track} + C_t^{risk}, \quad c_t = c_t^{track} + c_t^{risk} \\
F_t &= \begin{bmatrix} \mathrm{I}_2 & dt\mathrm{I}_2 \end{bmatrix}, \quad f_t = 0
\end{aligned} \tag{11}$$

In Eq. (3), $C_t^{track}$ is an identity matrix, and as analyzed above, the last two dimensions of the $C_t^{track}$ diagonal are replaced by $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$. Since $\beta_t^i$ and ($q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$) can be regarded as independent, Eq. (4) can still be multiplied by $C_t^{track}$. The components of $\boldsymbol{q_t^*}$ reflect the ego's attitude toward itself (tracking the reference trajectory) and the risk agents. We will present this attitude change in later explainability experiments.

### C. Backpropagation Loop

The previous subsection established the relationship between the feature vector and the dLQR parameters, enabling the fusion of a data-driven feature extraction network with a model-based dLQR planner into an end-to-end learnable framework. This network can adapt the planner's behavioral strategies based on environmental information to handle dynamically changing scenarios.

The dLQR planner obtains the optimal trajectory $\mathcal{P}$ with the current parameters by solving the LQR problem and closes to the expert trajectory ($\mathcal{P}^{target}, \mathcal{H}^{target}$) by imitation learning. The loss function consists of the $\texttt{MSE}(\cdot)$ of the trajectory points and the $\texttt{L1}(\cdot)$ error of the direction angles. The planner can only generate the coordinates of the trajectory points, and the heading angle $\mathcal{H}$ is calculated from the arctangent of the trajectory point difference.

$$loss = \texttt{Mean}\big(\texttt{MSE}(\mathcal{P}, \mathcal{P}^{target}) + \texttt{L1}(\mathcal{H}, \mathcal{H}^{target})\big) \tag{12}$$

The only part of the framework in this paper that requires training is the parameter $\theta$ of the feature vector extractor and the LSTM optimization layer. Based on the derivatives mentioned in III-B and the chain rule, the gradient descent loop of the framework can be easily obtained.

$$\frac{\partial loss}{\partial \theta} = \frac{\partial loss}{\partial \mathcal{W}} \frac{\partial \mathcal{W}}{\partial \boldsymbol{q}^*} \frac{\partial \boldsymbol{q}^*}{\partial \theta} \tag{13}$$

where the derivatives of the planner parameters with respect to the feature vector are from Eq. (11).

Algorithm 1 demonstrates the process of learning the data-driven and dLQR fusing planner by imitation from expert trajectories.

**Algorithm 1** Learning with *D2MFusion*

**Inputs**: rasterized BEV of six layers $\mathcal{X}$; lane centerline as reference trajectory $\tau_{1:T}^{ref}$; trajectories of agents $\tau_{1:T}^{1:N}$; initial state $p_1$; learning rate $\gamma$
Sample $n$ batches of $m$ data samples $(\mathcal{X}, \tau_{1:T}^{ref}, \tau_{1:T}^{1:N})$
**for** i=1 $\ldots n$ **do**
  $\boldsymbol{q} = \texttt{FeatureVecExtractor}(\mathcal{X})$ ▷ Eq. (9)
  $\boldsymbol{q}^* = \texttt{FeatureVecOpt}(\boldsymbol{q})$ ▷ Eq. (10)
  $\mathcal{W} = \texttt{FeatureVecTrans}(\boldsymbol{q}^*, \tau_{1:T}^{ref}, \tau_{1:T}^{1:N})$ ▷ Eq. (11)
  $\mathcal{P} = \texttt{LQR}(\mathcal{W}, p_1)$ ▷ iLQR [30]
  $\mathcal{H} = \texttt{arctan}\big(\texttt{Diff}(\mathcal{P})\big)$
  $loss = \texttt{Mean}\big(\texttt{MSE}(\mathcal{P}, \mathcal{P}^{target}) + \texttt{L1}(\mathcal{H}, \mathcal{H}^{target})\big)$ ▷ Eq. (12)
  $\theta \leftarrow \theta - \gamma \nabla loss$ ▷ Eq. (13)
**end for**

In Algorithm 1, the reference trajectory $\tau_{1:T}^{ref}$ is a cut-off part of the lane centerline with the speed limit of the current

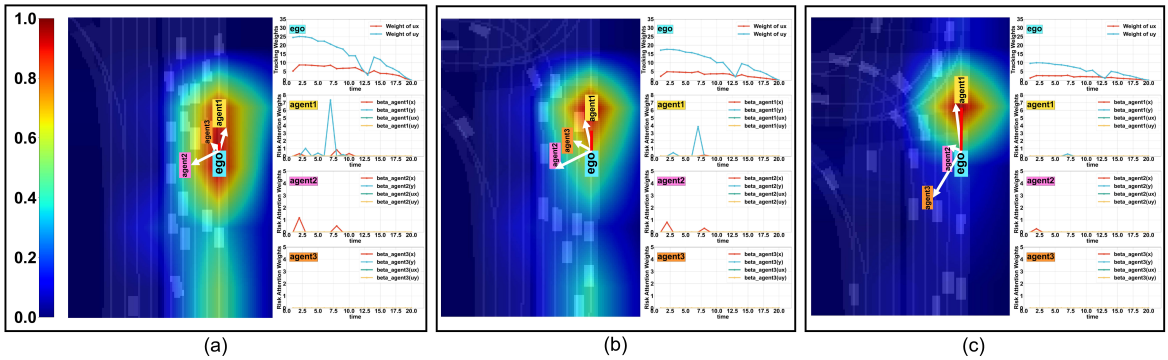


Fig. 2. **An explainable trace.** Each sub-figure displays the environmental feature map overlaid on the BEV and the trends of $\boldsymbol{q_t^*}$ components over the next $T$ time steps, including: tracking weights $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$ (cyan); attention weights of the agent ahead (yellow: $\beta_t^{1(1)} \sim \beta_t^{1(4)}$), the second closest agent (pink: $\beta_t^{2(1)} \sim \beta_t^{2(4)}$), and the third closest agent (orange: $\beta_t^{3(1)} \sim \beta_t^{3(4)}$). ($\boldsymbol{a}$) The ego follows the agent ahead closely. The environmental feature map focuses on the ego itself and the forward agent. The $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$, $\beta_t^{(1)}$, and $\beta_t^{(2)}$ of the agents are getting large. The ego adopts a safe strategy and slows down to avoid collision. ($\boldsymbol{b}$) The agent ahead accelerates away, and the area of interest moves with this agent. The $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$, $\beta_t^{(1)}$, and $\beta_t^{(2)}$ decrease. ($\boldsymbol{c}$) The area of concern moves out of the collision risk range with the forward agent. The $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$, $\beta_t^{(1)}$, and $\beta_t^{(2)}$ are reduced to a low level and the ego presents an aggressive strategy.

lane, and the trajectories of the agents $\tau_{1:T}^{1:N}$ are sampled from the real positions in the future time windows labeled in the dataset. In the real world, although we cannot obtain the future trajectories of the agents, the predicted future trajectories can be given based on uniform straight-line motions at their current speeds. All of the above trajectories are represented in the ego coordinate system. The iLQR method from [8], [30] is applied to solve the LQR problem.

## V. Experiments and Results

### A. Simulation Setup

**nuPlan dataset.** [31] is a large-scale autonomous driving benchmark. It includes 1200 hours of driving data from 4 cities. We choose nuPlan to validate our method as it contains complex reactive road conditions and dynamic scenarios that are highly suitable for our focus mentioned in I, and nuPlan is widely recognized for containing a large amount of high-quality data. We use scenarios in which the percentage of frames with agents in front of the ego is more than 70% for training and 200 scenarios for validation because we are more interested in how the ego deals when there is a reactive situation with other vehicles.

**Methods of evaluation.** nuPlan provides 14 low-level and 11 high-level metrics, and we evaluate our method and comparison cases using the following four metrics. *Percentage of Ego Collisions* measures the percentage of simulated scenarios in which the ego and other agents have collided. *Acceleration* is a statistic of the maximum absolute value of the acceleration of the ego for each scenario during the simulation. *Human Driving Similarity* is a statistic of the Euclidean distance between the planner's and the expert's trajectory points. *Goal Distance* is the distance between the ego's and the expert's final positions. All metrics are the lower, the better. It is worth mentioning that the four metrics aim for a balanced evaluation rather than simultaneous optimization. It is not easy to optimize the four metrics simultaneously; for example, a safe planner will cause an increase in acceleration, while a comfortable planner may lead to a deterioration of the other three metrics. We focus on safety and human driving similarity in dynamic environments. We use the closed-loop simulation, where our planner generates the trajectories of the ego, and the trajectories of the agents are given by the labels in the dataset, synchronized in time. In the comparison experiments, the data are averaged over the validation set.

**Comparison cases.** Comparison experiments are performed for six planners. *Ours* refers to the method of this paper; *Human* refers to the human driving data in the dataset; *CNN* refers to the data-driven method only, where the BEV image directly controls the trajectory; *C-LQR(HO): Constant LQR (human optimized)* refers to the model-based planner only, with parameters in Eq. (11) that are empirically optimized by humans and are constant; *C-LQR (LD): Constant LQR (learned from the dataset)* also refers to the model-based planner only, but with parameters that are obtained by learning from the dataset and are constant; and *UrbanDriver* refers to the baseline of nuPlan [32]. The same CNN backbone extracts features for *Ours* and *CNN*. For *CNN*, the BEV image is the input; the resnet50 extracts the features and then passes through an FC layer to output a $3 \times T$ dimensional vector, which is transformed to obtain the trajectory. For *C-LQR(HO)* and *C-LQR(LD)*, only a part of the lane centerline is cut off with the lane speed limit as the reference trajectory, and no additional information from the scenario is used. For *UrbanDriver*, the ego poly, the agent poly, and the map polys are forwarded to the PointNet layers. The features go through a multi-head attention layer, which considers their relations to output the trajectory of the ego.

### B. Simulation Results and Discussion

**D2MFusion learns to extract an explainable feature vector that promotes safe behaviors.** In the simulation, the ego considers the three closest agents, assigning higher attention to closer ones. An agent within 30 meters ahead in the same lane is considered the most critical. The atten-

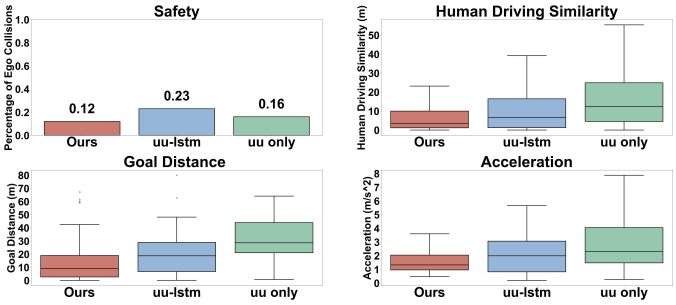


Fig. 3. **The relationship between the feature vector and dLQR parameters.** The hidden feature vector $\boldsymbol{q}$ used to control only the tracking term increases safety but decreases comfort and human driving similarity. Adding LSTM to the tracking term increases comfort but lowers safety. The feature vector used for tracking and risk terms after LSTM can be advanced in safety, human driving similarity, goal distance, and comfort.

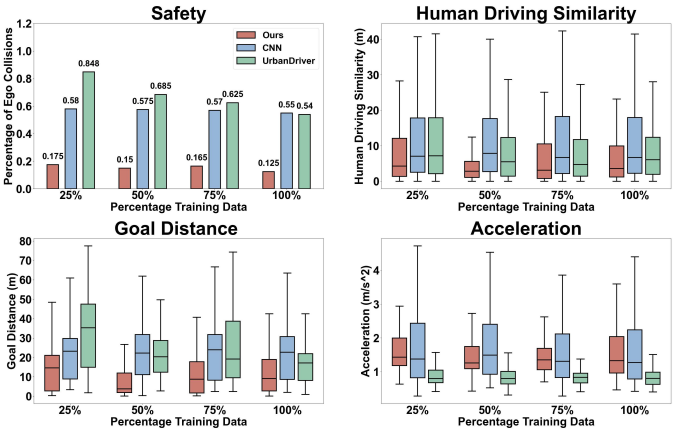


Fig. 4. **Model data efficiency study.** Model performance trained at 25%, 50%, 75%, and 100% training data. Our approach is able to achieve relatively high safety and human driving similarity and low goal distance with a small percentage of training data.

tion mechanism relies on distance rather than lane priority, so closer agents, regardless of lane, naturally receive more focus. This design suits scenarios such as successive lane changes and robot navigation. Fig. 2 shows three time steps of the ego's low-speed following. Each sub-figure overlays the environmental feature map extracted by the CNN's last layer on the BEV image, with warmer colors indicating higher attention. On the right, trends of the components of $\boldsymbol{q_t^*}$ over the next $T$ time steps are shown: tracking weights $q_t^*(u_{xt}^2)$ and $q_t^*(u_{yt}^2)$ (cyan), and attention weights of the agent ahead (yellow), second (pink), and third (orange) closest agents, corresponding to the colors marking the agents. Analyzing $\boldsymbol{q_t^*}$ reveals the planner behavior: larger tracking weights lead to smaller control inputs and more conservative trajectories. Higher $\beta_t^{(1)}$ and $\beta_t^{(2)}$ indicate a stronger avoidance of the respective agents. The smaller $\beta_t^{(3)}$ and $\beta_t^{(4)}$ increase the terms $q_t^*(u_{xt}^2)(1-\beta_t^{(3)})$ and $q_t^*(u_{yt}^2)(1-\beta_t^{(4)})$ (Eq. 11), further enhancing the conservativeness. In summary, larger $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$, $\beta_t^{(1)}$, $\beta_t^{(2)}$ and smaller $\beta_t^{(3)}$, $\beta_t^{(4)}$ correspond to more cautious behavior and safer trajectories. In Fig. 2(a), the ego closely follows the leading agent, with attention focused on both. Increased $q_t^*(u_{xt}^2)$, $q_t^*(u_{yt}^2)$, $\beta_t^{(1)}$, and $\beta_t^{(2)}$ reflect a safe strategy of slowing down to avoid collision. By Fig. 2(b), as the lead agent accelerates and the gap widens, these weights decrease, and attention shifts forward. In Fig. 2(c), the lead agent leaves the risk zone, the focus is on it, and the lower weights indicate that the ego transitions to an aggressive strategy. The trace in Fig. 2 shows that the feature vector $\boldsymbol{q*}$ combined into dLQR is explainable. The planner can adjust its attitude towards itself and different agents and switch reactive strategies in real-time according to the dynamically changing environmental features, thus realizing safe reactive navigation.

**The feature vector goes through the LSTM layer and is transformed into the tracking and risk parameters of dLQR, contributing to improved performance.** Section IV-B describes how the feature vector is mapped into two sets of parameters: the weights of the control quadratic terms in the tracking part and the weights of the agents in the risk part. Fig. 3 reports the effect of different optimization and transformation strategies for the hidden feature vector, all trained with the same CNN backbone and dataset. In Fig. 3, *Ours* denotes

TABLE I
PERFORMANCE COMPARISON

| Model | %Collision Percentage | Acceleration |
|---|---|---|
| | mean ± std | mean ± std |
| Human | n/a | 0.95 ± 0.30 |
| CNN | 55.0% | 1.68 ± 1.10 |
| C-LQR(HO) | 33.5% | 2.19 ± 1.71 |
| C-LQR(LD) | 26.0% | 3.09 ± 7.69 |
| UrbanDriver | 54.0% | **0.84 ± 0.29** |
| Ours | **12.5%** | 1.94 ± 1.70 |
| **Model** | **Human Driving Sim.** | **Goal Distance** |
| | **mean ± std** | **mean ± std** |
| Human | n/a | n/a |
| CNN | 9.61 ± 9.46 | 21.17 ± 14.21 |
| C-LQR(HO) | 8.31 ± 7.60 | 18.31 ± 14.18 |
| C-LQR(LD) | 5.39 ± 5.44 | 12.21 ± 12.24 |
| UrbanDriver | 7.54 ± 6.60 | 16.21 ± 9.09 |
| Ours | **5.10 ± 5.33** | **12.08 ± 10.62** |

transforming the LSTM output $\boldsymbol{q}^*$ into both tracking and risk parameters. *uu only* directly maps the hidden feature vector $\boldsymbol{q}$ to the tracking parameters, modifying only the weights of the control quadratic terms. *uu-lstm* maps the LSTM output $\boldsymbol{q}^*$ to the tracking parameters only. The $\beta$ *only* method fails to converge, as the planner cannot generate safe trajectories from high-speed references when no leading agent is present. The results of *uu only* indicate that adjusting only the weights of the control quadratic terms can yield safe trajectories, since the control magnitude influences the trajectory length. However, the weights across different time windows are independent, leading to large accelerations and reduced performance in goal distance and human driving similarity. *uu-lstm* shows improved comfort relative to *uu only*, but neglecting agent-related weights and limiting acceleration/deceleration reduces safety. By contrast, *Ours* leverages LSTM outputs for the weights of both the tracking and risk components, achieving superior results in safety, human driving similarity, goal distance, and comfort.

**D2MFusion strikes a well-balanced behavior with an emphasis on safety.** Table I shows the performance of each comparison method for the closed-loop simulation on the

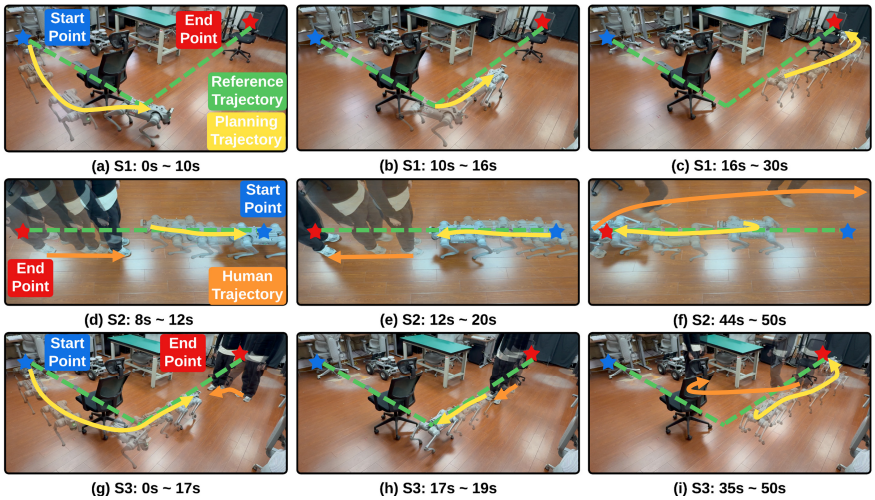


Fig. 5. **Video snapshots of our method in three real-world experiments.** Scenario I in which Go2 bypasses two obstacles that appear on the reference trajectory to reach the endpoint safely is illustrated from (a) to (c). Go2 interacts with the human and always maintains a safe distance, as shown in (d)∼(f). In Scenario III, Go2 bypasses the first obstacle and then interacts safely with the human who suddenly appears, and finally bypasses the second obstacle to reach the endpoint as shown in (g)∼(i).

validation set. The metrics are expressed as the mean value with a standard deviation appended. Our model works best with safety, human driving similarity, and target reaching. The safety improvement is coupled with an increase in acceleration. *UrbanDriver* performs best in comfort, but its low acceleration results in unsatisfactory performance in other metrics. The results of *C-LQR(HO)* and *C-LQR(LD)* indicate that the parameters obtained by learning from the dataset perform better than the human-optimized ones. In comparison with *Ours*, the use of constant parameters in both of them lacks adaptability in dealing with dynamically changing environments. *It is worth noting that our approach uses the neural planner to generate the general reactive behavior, and there is usually a post-processor to ensure absolute safety, which, for comparison purposes, we do not include in this work.*

**D2MFusion demonstrates high data efficiency.** Due to the integration of the dLQR, our framework already has a simple kinematic model and the ability to follow a reference trajectory without training. Our model can improve data efficiency compared with other methods. In Fig. 4, the metrics of the three models based on learning in the comparison cases are shown on four-size datasets. Training is performed on datasets of different sizes, and simulation is on the same validation set. The results show that our method has better human driving similarity, safety, and goal distance, even with a small dataset for training. Although the acceleration metric is not minimized, the smaller acceleration of the other models leads to continuous over or under-speeding, which is reflected in the other metrics by increasing the probability of collision, off-target, and a decrease in similarity to humans.

### C. Real World

**Testing Environment.** We deploy D2MFusion on a Unitree Go2 robot for real-world experiments, as illustrated in Fig. 5. Three navigation scenarios are designed in the laboratory. Scenario I specifies a reference trajectory of 2.7m forward followed by 4m to the left, with two stationary obstacles placed on the trajectory. Scenario II involves a 2.8m forward trajectory, during which a human interacts with the robot. Scenario III shares the reference trajectory of Scenario I, but includes two stationary obstacles and a human appearing near the trajectory to interact with the robot.

**Implementation.** We use ROS2 for control and sensor communication. Expert trajectories are collected by manually operating the Go2 with a joystick in the testing environment, including five obstacle avoidance trajectories (732 points) and three human interaction trajectories (539 points). The dataset contains robot and obstacle states (position, orientation, velocity) captured by the NOKOV motion capture system at 10Hz. Training is performed on an NVIDIA RTX4060 (8GB). During both training and experiments, the input BEV representation has four channels, excluding the lane and road layers used in the simulation. Each BEV layer is generated from motion capture data and transformed into the Go2 coordinate frame, consistent with the simulation setup. The BEV covers 2.5m$\times$ 2.5m with a resolution of 0.05m. The planned trajectory from D2MFusion is executed by Go2 through its low-level tracking controller. Data processing and trajectory planning run on a computer with an Intel Core i9-13900HX CPU and 32GB RAM. The planner operates at 10Hz with a 2s horizon, identical to the simulation setting. Fig. 5 illustrates the start and end points (blue and red stars), the reference trajectory (green dashed line), the trajectory planned by D2MFusion (yellow arrows), and the human trajectory (orange arrows).

**Scenario I:** From $0s$ to $10s$ (Fig. 5(a)), a stationary obstacle appears on the robot's forward route. The planner generates a trajectory to circumvent it. After passing the obstacle, the robot continues along the reference trajectory (Fig. 5(b)). From $16s$ to $30s$, another obstacle appears, causing the planner to generate a detour, and the robot stops at the endpoint (Fig. 5(c)). **Scenario II:** Initially, there are no obstacles. The robot follows the reference trajectory until it encounters a

standing human, triggering a stop plan. From $8s$ to $12s$, as the human approaches, the planner adjusts to move backward, maintaining a safe distance (Fig. 5(d)). Once the human steps back, the robot resumes forward motion (Fig. 5(e)). From $44s$ to $50s$, the human bypasses the robot, which tends to back away first and then follows the reference trajectory to the endpoint (Fig. 5(f)). **Scenario III:** From $0s$ to $17s$, the robot circumvents an obstacle and turns left. A human suddenly appears, activating the planner to stop the robot (Fig. 5(g)). During repeated interactions, the robot maintains a safe distance (Fig. 5(h)). From $35s$ to $50s$, after the human leaves, the robot avoids the human and then circumvents the obstacle to reach the endpoint (Fig. 5(i)). Experiments on Go2 demonstrate that our method can react safely in dynamic environment navigation.

## VI. Conclusion

In this work, we introduced D2MFusion, a novel differentiable planning architecture that adopts an embedding feature vector reflecting the ego's reaction attitude to the dynamic agents. This neural planner can adjust the planning strategies in the planning process to realize safe reactive navigation. Additionally, we proposed a fusing method of integrating neural networks and differentiable optimizers by incorporating LQR into the backpropagation process, facilitating end-to-end learning and improving data efficiency. The experiments show that this neural planner is highly explainable and data-efficient. In comparison experiments, the method performs well in safety, target reaching, and human driving similarity. Finally, our experiments on a real robot indicate that our method is effective and can be applied to reactive navigation in generic dynamic environments.

## References


[1] A. Suresh, A. Taylor, L. D. Riek, and S. Martínez, "Robot navigation in risky, crowded environments: Understanding human preferences," *IEEE Robot. Automat. Lett.*, vol. 8, no. 9, pp. 5632–5639, 2023.

[2] A. Taylor and L. D. Riek, "Regroup: A robot-centric group detection and tracking system," in *Proc. ACM/IEEE Int. Conf. Hum.-Robot Interact.*, 2022, pp. 412–421.

[3] Y. Xu, J. Xie, T. Zhao, C. Baker, Y. Zhao, and Y. N. Wu, "Energy-based continuous inverse optimal control," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 34, no. 12, pp. 10 563–10 577, 2023.

[4] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun, "Jointly learnable behavior and trajectory planning for self-driving vehicles," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, p. 3949–3956.

[5] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural mpc: Deep learning model predictive control for quadrotors and agile robotic platforms," *IEEE Robot. Automat. Lett.*, vol. 8, no. 4, pp. 2397–2404, 2023.

[6] J. Liu, S. Lyu, D. Hadjivelichkov, V. Modugno, and D. Kanoulas, "Vit-a*: Legged robot path planning using vision transformer a*," 2023.

[7] A. Hu, Z. Murez, N. Mohan, S. Dudas, J. Hawke, V. Badrinarayanan, R. Cipolla, and A. Kendall, "Fiery: Future instance prediction in bird's-eye view from surround monocular cameras," in *Proc. IEEE Int. Conf. Comp. Vis.*, 2021, pp. 15 253–15 262.

[8] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proc. Int. Conf. Machine Learning*, vol. 70, 2017, pp. 136–145.

[9] W. Farag and Z. Saleh, "Behavior cloning for autonomous driving using convolutional neural networks," in *Proc. Int. Conf. Innov. Intell. Informat., Comput., Technol.*, 2018, pp. 1–7.

[10] Z. Huang, C. Lv, Y. Xing, and J. Wu, "Multi-modal sensor fusion-based deep neural network for end-to-end autonomous driving with scene understanding," *IEEE Sens. J.*, vol. 21, no. 10, pp. 11 781–11 790, 2021.

[11] L. Chen, L. Platinsky, S. Speichert, B. Osiński, O. Scheel, Y. Ye, H. Grimmett, L. Del Pero, and P. Ondruska, "What data do we need for training an av motion planner?" in *Proc. Int. Conf. Robot. Automat.*, 2021, pp. 1066–1072.

[12] J. Ngiam, B. Caine, V. Vasudevan, Z. Zhang, H. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, D. Weiss, B. Sapp, Z. Chen, and J. Shlens, "Scene transformer: A unified multi-task model for behavior prediction and planning," *CoRR*, vol. abs/2106.08417, 2021.

[13] Z. Wen, Y. Zhang, X. Chen, and J. Wang, "Tofg: A unified and fine-grained environment representation in autonomous driving," in *Proc. Int. Conf. Robot. Automat.*, 2023, pp. 1565–1571.

[14] V. D. Sharma, L. Zhou, and P. Tokekar, "D2coplan: A differentiable decentralized planner for multi-robot coverage," in *Proc. Int. Conf. Robot. Automat.*, 2023, pp. 3425–3431.

[15] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger, "Exploring data aggregation in policy learning for vision-based urban autonomous driving," in *Proc. IEEE Conf. Comp. Vis. Pattern Recog.*, 2020, pp. 11 760–11 770.

[16] S. Jiwani, X. Li, S. Karaman, and D. Rus, "Risk-aware neural navigation from bev input for interactive driving," in *Proc. Int. Conf. Robot. Automat.*, 2023, pp. 5659–5665.

[17] R. Yonetani, T. Taniai, M. Barekatain, M. Nishimura, and A. Kanezaki, "Path planning using neural a* search," *CoRR*, vol. abs/2009.07476, 2020.

[18] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *IEEE Trans. Robotics*, vol. 39, no. 3, pp. 1749–1767, 2023.

[19] H. Zhu, F. M. Claramunt, B. Brito, and J. Alonso-Mora, "Learning interaction-aware trajectory predictions for decentralized multi-robot motion planning in dynamic environments," *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 2256–2263, 2021.

[20] J. Zhou, R. Wang, X. Liu, Y. Jiang, S. Jiang, J. Tao, J. Miao, and S. Song, "Exploring imitation learning for autonomous driving with feedback synthesizer and differentiable rasterization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1450–1457.

[21] M. Selim, A. Alanwar, S. Kousik, G. Gao, M. Pavone, and K. H. Johansson, "Safe reinforcement learning using black-box reachability analysis," *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, p. 10665–10672, 2022.

[22] P. Karkus, X. Ma, D. Hsu, L. Kaelbling, W. S. Lee, and T. Lozano-Perez, "Differentiable algorithm networks for composable robot learning," in *Proc. Robotics: Sci. Syst.*, FreiburgimBreisgau, Germany, June 2019.

[23] P. Karkus, B. Ivanovic, S. Mannor, and M. Pavone, "Diffstack: A differentiable and modular control stack for autonomous vehicles," in *Proc. Conf. Robot. Learn.*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252752748

[24] X. Li, I. Gilitschenski, G. Rosman, S. Karaman, and D. Rus, "Multi-abstractive neural controller: An efficient hierarchical control architecture for interactive driving," *IEEE Robot. Automat. Lett.*, vol. 8, no. 8, pp. 4737–4744, 2023.

[25] Z. Huang, H. Liu, J. Wu, and C. Lv, "Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving," *IEEE Trans. Neural Networks Learn. Syst.*, pp. 1–15, 2023.

[26] F. Yang, C. Wang, C. Cadena, and M. Hutter, "iPlanner: Imperative Path Planning," in *Proc. Robotics: Sci. Syst.*, Daegu, Republic of Korea, July 2023.

[27] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016.

[28] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," *CoRR*, vol. abs/1810.13400, 2018.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[30] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 4906–4913.

[31] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. M. Wolff, A. H. Lang, L. Fletcher, O. Beijbom, and S. Omari, "nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," *CoRR*, vol. abs/2106.11810, 2021.

[32] O. Scheel, L. Bergamini, M. Wolczyk, B. Osinski, and P. Ondruska, "Urban driver: Learning to drive from real-world demonstrations using policy gradients," *CoRR*, vol. abs/2109.13333, 2021.