Jointly Learnable Behavior and Trajectory Planning for Self-Driving Vehicles

Abbas Sadat*,1, Mengye Ren*,1,2, Andrei Pokrovsky[3], Yen-Chen Lin[4], Ersin Yumer[1], Raquel Urtasun[1,2]

***Abstract*— The motion planners used in self-driving vehicles need to generate trajectories that are safe, comfortable, and obey the traffic rules. This is usually achieved by two modules: *behavior planner*, which handles high-level decisions and produces a coarse trajectory, and *trajectory planner* that generates a smooth, feasible trajectory for the duration of the planning horizon. These planners, however, are typically developed separately, and changes in the behavior planner might affect the trajectory planner in unexpected ways. Furthermore, the final trajectory outputted by the trajectory planner might differ significantly from the one generated by the behavior planner, as they do not share the same objective. In this paper, we propose a jointly learnable behavior and trajectory planner. Unlike most existing learnable motion planners that address either only behavior planning, or use an uninterpretable neural network to represent the entire logic from sensors to driving commands, our approach features an interpretable cost function on top of perception, prediction and vehicle dynamics, and a joint learning algorithm that learns a shared cost function employed by our behavior and trajectory components. Experiments on real-world self-driving data demonstrate that jointly learned planner performs significantly better in terms of both similarity to human driving and other safety metrics, compared to baselines that do not adopt joint behavior and trajectory learning.**

I. Introduction

Modern motion planners used in today's self-driving vehicles (SDVs) are typically composed of two distinct modules. The first module, referred to as *behavior planning*, is responsible for providing high-level decisions given the output of perception and prediction (*i.e.* perception outputs extrapolated to future timestamps). To name a few, examples of such decisions are lane changes, turns and yields at an intersection. The second module, referred to as *trajectory planning*, takes the decision of the behavior planner and a coarse trajectory and produces a smooth trajectory for the duration of the planning horizon (typically 5 to 10s into the future). This is then passed to the control module to execute the maneuver.

The role of the behavior planner is to constrain the trajectory generation such that a high-level objective is achieved. Earlier planners based on simple rule-based behavior selection or finite state machines are unfortunately unable to handle decision making in complex real-world urban scenarios. Alternative approaches try to optimize a behavioral objective by using sequential A* search [1] or parallel sampling methods [2], while reasoning about traffic-rules and other actors. A popular approach to alleviate the gap between behavior and trajectory planning is to restrict the motion of the SDV to a path (*e.g.* , lane centerline) and find the velocity profile that optimizes the behavioral objective.

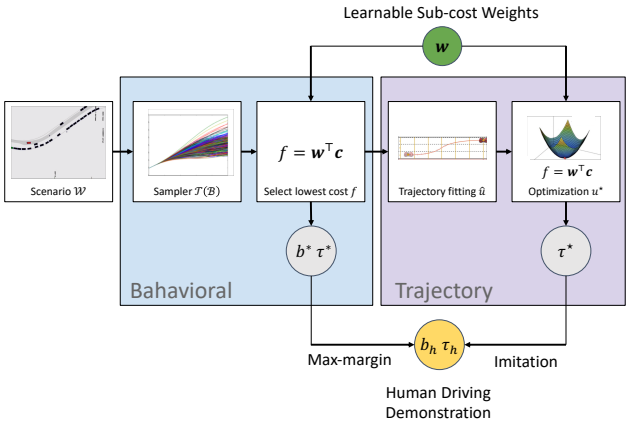

Fig. 1: Our learnable motion planner has discrete and continuous components, minimizing the same cost function with a same set of learned cost weights.

Trajectory planning is typically formulated as an optimization problem where a low-level objective is optimized locally to satisfy both the high-level decisions as well as the kinematics and dynamics constraints [3]. Many different continuous solvers, such as iLQR and SQP, have been exploited to solve the optimization problem. An alternative approach is sampling, where trajectories are generated in a local region defined by the high-level decisions, and the trajectory with lowest cost is selected for execution. Searching through a spatio-temporal state-lattice representing continuous trajectories is another popular approach.

While great progress has been achieved in the development of individual methods for either behavior or trajectory planners, little effort has been dedicated to jointly designing these two modules. As a consequence, they typically do not share the same cost function, and thus changes in the behavior planner can have negative effects on the heavily tuned trajectory planner. Furthermore, the gains and costs of these planners are mostly manually tuned. As a result, motion planning engineers spend a very significant amount of their development time re-tuning and re-designing the planners given changes in the stack.

* Equal contribution

[1]Abbas Sadat, Mengye Ren, Ersin Yumer and Raquel Urtasun are with Uber Advanced Technologies Group, 661 University Avenue, Suite 720, Toronto, Ontario, Canada, M5G 1M1. Email: {asadat,mren3,yumer,urtasun}@uber.com.

[2]Mengye Ren and Raquel Urtasun are also with University of Toronto.

[3]Andrei Pokrovsky is with GraphCore. Work done at Uber.

[4]Yen-Chen Lin is with Massachusetts Institute of Technology. Work done at Uber.

In this paper we tackle this issue by designing a motion planner where both the behavior and the trajectory planners share the same cost function. Importantly, our planner can be trained jointly end-to-end without requiring manual tuning of the costs functions. Our motion planner is designed to produce comfortable, safe, and mission-oriented trajectories and is able to handle complex traffic scenarios such as nudging to bicyclists and objects that partially occupy lanes, and yielding at intersections.

We demonstrate the effectiveness of our approach in real-world urban scenarios by comparing the planned trajectories to comfortable trajectories performed by cautious human drivers. We show that our planner can learn to produce comfortable trajectories in terms of jerk and stay more close to the manually-driven trajectories when compared to the baselines. Additionally, we demonstrate the performance of our planner on a very large-scale dataset of highly challenging safety critical scenarios.

II. Related Work

a) Behavior Planning: Bender *et al.* [3] propose to enumerate behaviors such that each discrete class of behaviors could later be considered in an independent trajectory optimization step. Gu *et al.* [4] propose a multi-phase decision making framework where a traffic-free reference planning over a long range is performed first, followed by a traffic-based refinement where other actors are taken into account; lastly, a final step of local trajectory planning addresses the short-term motion horizon in a refined manner. In [1], a search-based behavior planning approach that is capable of handling hundreds of variants in real-time was proposed, addressing the limitation of [3]. This is achieved by representing the search space and the driving constraints with a geometric representation that is amenable to modeling predictive control schemes, and using an explicit cost-to-go map. Recently, Fan *et al.* [5] proposed a framework that uses dynamic programming to find an approximate path and a speed profile iteratively in an EM-like scheme, followed by quadratic programming optimization of the cost function. In all of the above cases, manually designed costs are used that consider lane boundaries, collision, traffic lights, and other driving conditions. Even though hand tuning the contribution of each constraint is possible, it is very time consuming.

b) Trajectory Planning: Werling *et al.* [2] introduced a combinatorial approach to trajectory planning, where a set of candidate swerve trajectories align the vehicle to the center of a lane given by an upstream behavior planning. Similarly, approaches with combinatorial schemes and lattices were used in [6], [7], [8]. Conversely, discretization is avoided in [9], by introducing a continuous non-linear optimization method where obstacles in the form of polygons are converted to quadratic constraints.

c) Learned Motion Planning: Learning approaches to motion planning have mainly been studied from an imitation learning (IL) [10], [11], [12], [13], [14], [15], [16], or reinforcement learning (RL) [17], [18], [19], [20] perspective. While most IL approaches provide an end-to-end training framework to control outputs from sensory data, they suffer from compounding errors due to the sequential decision making process of self-driving. Moreover, these approaches require significantly more data due to the size of learnable parameters in modern networks. Fan *et al.* [5] propose a ranked based IL framework for learning the reward function based on a linear combination of features. One major difference from our approach is that their continuous optimizer is not jointly learned. The success of RL approaches to date has been limited to only simulated environments or simple problems in robotics. More importantly, both IL and RL approaches, in contrast to the traditional motion planners, are not interpretable. Recently, Zeng *et al.* [21] introduced an end-to-end neural planner where sensor data is processed upto the end of a behavioral planner cost function, in a deep network together with perception and prediction outputs to increase interpretability of such approaches. Ratliff *et al.* [11] use maximum margin for behavioral planning. In contrast, we develop a framework where we tackle both behavior planning and local trajectory planning with a shared cost function that can be learned end-to-end.

The contribution of our work can be viewed as a combination of the advantages from both learned and traditional two-stage approaches: 1) Like the learned approaches, we help eliminate the time-consuming, error-prone, and iterative hand-tuning of the gains of the planner costs. 2) Unlike the learned approaches above, we do so within a framework of interpretable costs jointly imposed on these modules. Therefore, even though our motion planner is data-driven, it still uses the widely adapted, interpretable costing concepts for each driving constraint. Moreover, it is also end-to-end trainable.

III. Joint Behavior-Trajectory Planner

Motion planners of modern self-driving cars are composed of two modules. The behavioral planner is responsible for making high level decisions. The trajectory planner takes the decision of the behavioral planner and a coarse trajectory and produces a smooth trajectory for the duration of the planning horizon. Unfortunately these planners are typically developed separately, and changes in the behavioral planner might affect, in unexpected ways, the trajectory planner. Furthermore, the trajectory outputted by the trajectory planner might differ significantly in terms of behavior from the one returned by the behavioral planner as they do not share the same objective. To address this issue, in this paper we propose a novel motion planner where both the behavioral and trajectory planners share the same objective.

We use $\mathcal{W}$ to denote the input to the motion planner from the upstream modules at each planning iteration. In particular, $\mathcal{W}$ includes the desired route as well as the state of the world, which contains the SDV state, the map, and the detected objects. Additionally, for each object, multiple future trajectories are predicted including their probabilities. The planner outputs a high-level behavior b and a trajectory τ that can be executed by the SDV for the planning horizon $T = 10s$. Here we define behavior as a driving-path that

the SDV should ideally converge to and follow. These paths are obtained by considering *keep-lane*, *left-lane-change*, and *right-lane-change* maneuvers. We refer the reader to Fig. 2-A for an illustration. At each planning iteration, depending on the SDV location on the map, a subset of these behaviors, denoted by $\mathcal{B}(\mathcal{W})$, is allowed by traffic-rules and hence considered for evaluation. We then generate low-level realizations of the high-level behaviors by generating a set of trajectories $\mathcal{T}(b)$ relative to these paths (see Section (V-A)). Assuming the SDV follows a bicycle model, we can represent the vehicle state at time t by $X_t = [\mathbf{x}_t, \theta_t, \kappa_t, v_t, a_t, \dot{\kappa}_t]$. Here $\mathbf{x}$ is the Cartesian coordinate of position; θ is the heading angle; κ is the curvature; v is the velocity; a is the acceleration; and $\dot{\kappa}$ is the twist (derivative of curvature). A trajectory τ is defined as a sequence of vehicle states at discrete time steps ahead.

The objective of the planner is then to find a behavior and a trajectory that is safe, comfortable, and progressing along the route. We find such behavior and trajectory by minimizing a cost function that describes the desired output:

$$b^*, \tau^* = \underset{b \in \mathcal{B}(\mathcal{W}), \tau \in \mathcal{T}(b)}{\operatorname{argmin}} f(\tau, b, \mathcal{W}; \mathbf{w}) \tag{1}$$

We next describe the costs in more details, followed by our inference and learning algorithms.

IV. A Unified Cost Function

In this section we describe our unified cost function for our behavioral and trajectory planners. Given the sets of candidate behaviors and trajectories, the cost function f is used to choose the best (b, τ). The cost function consists of sub-costs $\mathbf{c}$ that focus on different aspects of the trajectories such as safety, comfort, feasibility, mission completion, and traffic rules. We thus define

$$f(\tau, b, \mathcal{W}; \mathbf{w}) = \mathbf{w}^\top \mathbf{c}(\tau, b, \mathcal{W}). \tag{2}$$

where the weight vector $\mathbf{w}$ captures the importance of each sub-cost. The following sub-sections introduce $\mathbf{c}$ in detail.

A. Obstacle

A safe trajectory for the SDV should not only be collision-free, but also satisfy a safety-distance to the surrounding obstacles, including both the static and dynamic objects such as vehicles, pedestrians, cyclists, unknown objects, *etc.* Here we use c_{overlap} and c_{obstacle} to capture the spatio-temporal overlap and violation of safety-distance respectively. For this, the SDV polygon is approximated by a set of circles with the same radii along the vehicle, and we use the distance from the center of the circles to the object polygon to evaluate the cost (see Fig. 2-C). The overlap cost c_{overlap} is then 1 if a trajectory violates the spatial occupancy of any obstacle in a given predicted trajectory, and is averaged across all possible predicted trajectories weighted by their probabilities. The obstacle cost c_{obstacle} penalizes the squared distance of the violation of the safety-distance d_{safe}. This cost is scaled by the speed of the SDV, making the distance violation more costly at higher speeds. This also prevents accumulating cost in a stopped trajectory when other actors get too close to the SDV.

B. Driving-path and lane boundary

The SDV is expected to adhere to the structure of the road, *i.e.* , it should not go out of the lane boundary and should stay close to the center of the lane. Therefore, we introduce sub-costs that measure such violations. The driving-path and boundaries that are considered for these sub-costs depend on the candidate behavior (see Fig. 2-B). The driving-path cost c_{path} is the squared distance towards the driving path (red dotted lines in Fig. 2-B). The lane boundary cost c_{lane} is the squared violation distance of a safety threshold.

C. Headway

As the SDV is driving behind a leading vehicle in either lane-following or lane-change behavior, it should keep a safe longitudinal distance that depends on the speed of the SDV and the leading vehicle. We compute the headway cost as the violation of the safety distance after applying a comfortable constant deceleration, assuming that the leading vehicle applies a hard brake [22]. To compute the cost above, we need to decide which vehicles are leading the SDV at each time-step in the planning horizon. A possible approach is to associate vehicles to lanes based on distance to the center-line. However, this approach can be too conservative and make nudging behavior difficult. Instead, we use a weight function of the lateral distance between the SDV and other vehicles to determine how relevant they are for the headway cost (see Fig. 3). Hence, the distance violation costs incurred by vehicles that are laterally aligned with the SDV dominate the cost. This is also compatible with lane change manoeuvres where deciding the lead vehicles can be difficult.

D. Yield

Pedestrians are vulnerable road users and hence require extra caution. When a pedestrian is predicted to be close to the boundary of the SDV lane or crossing it, we impose a stopping point at a safe longitudinal distance and penalize any trajectory that violates it (see Figure 2-D). This is different from a simple Cartesian distance as it does not allow going around the pedestrians in order to progress in the route. The yield cost c_{yield} penalizes the squared longitudinal violation distance weighted by the pedestrian prediction probability. Similarly, the SDV needs to keep a safe longitudinal distance to vehicles that are predicted to be crossing an intersection, as well as stop at signal-controlled intersections. We use the same cost form as the pedestrian cost, but with different safety margins.

E. Route

The mission route is represented as a sequence of lanes, from which we can specify all lanes that are on the route or are connected to the route by permitted lane-changes. A behavior is desirable if the goal lane is closer to the route than the current lane. Therefore, we penalize the number

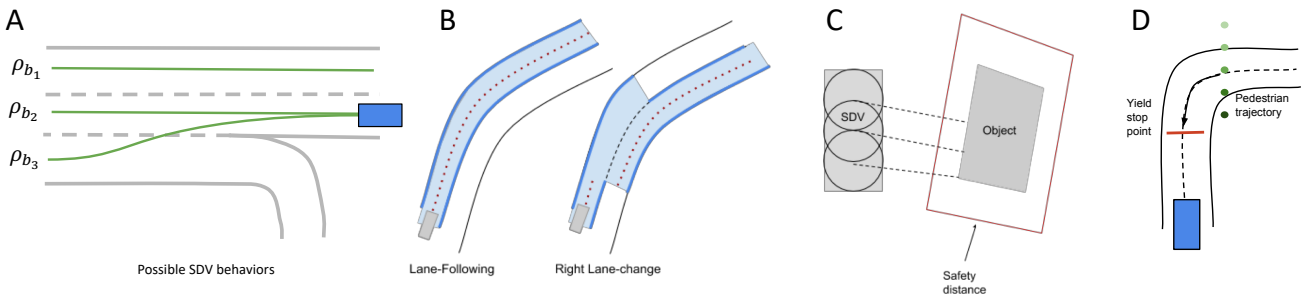

Fig. 2: A: Given a scenario, we generate a set of possible SDV behaviors. B: Left and right lane boundaries and the driving path that are relevant to the intended behavior are considered in the cost function. C: SDV geometry for spatiotemporal overlapping cost are approximated using circles. D: The SDV yields to pedestrians through stop lines on the driving paths.

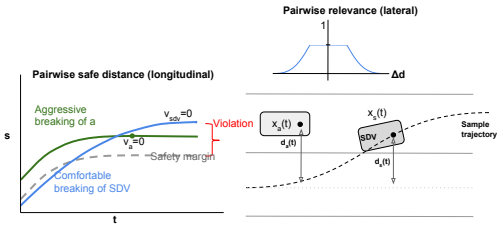

Fig. 3: Left: Headway cost penalizes unsafe distance to leading vehicles. Right: for each sampled trajectory, a weight function determines how relevant an obstacle is to the SDV in terms of its lateral offset.

of lane-changes that is required to converge to the route. Furthermore, violation of a distance-threshold to the end of a lane is penalized to force lane-changes from dead-end lanes to lanes that the SDV can continue on the route.

F. Cost-to-go

The sub-costs introduced so far evaluate the trajectory within the planning horizon, ignoring what comes beyond it. Additionally, we incorporate a cost-to-go function to capture the value of the final state of the SDV in a trajectory. This can prevent the planner to choose actions that are sub-optimal beyond the horizon or, worst, take the SDV into an inevitable unsafe situation. For this purpose, we compute the deceleration needed for slowing-down to possible up-comming speed-limits and use the square of the violation of the comfortable deceleration as cost-to-go. Consequently, trajectories that end with high velocity close to turns or stop-signs will be penalized.

G. Speed limit, travel distance and dynamics

Using the speed-limit of a lane, which is available in the map data, we introduce a cost that penalizes a trajectory if it goes above the eligible speed. The speed limit cost c_{speed} is the squared violation in speed. In order to favor trajectories that advance in the route, we use the travelled longitudinal distance as a reward. Since the SDV is physically limited to certain ranges of acceleration, curvature, *etc*, we prune trajectories that violate such constraints. Additionally, we introduce costs that penalize aggressive motions to promote comfortable driving. Specifically, the dynamics cost c_{dyn} consists of the squared values of jerk and violation thereof, acceleration and violation thereof, lateral acceleration and violation thereof, lateral jerk and violation thereof, curvature, twist, and wrench.

Algorithm 1 Inference of our joint planner

procedure INFERENCE($\mathbf{w}, \mathcal{W}$)
  ▷ The behavioral planner
  $\tau^*, b^* \leftarrow \text{argmin}_{b\in\mathcal{B}, \tau\in\mathcal{T}(b)} f(\tau, b, \mathcal{W}; \mathbf{w})$
  $u \leftarrow$ TRAJECTORYFITTER(τ^*, b^*)
  ▷ The trajectory planner
  while u not converge **do**
    $u \leftarrow$ OPTIMIZERSTEP($f(\tau^{(T)}(u), b^*, \mathcal{W}; \mathbf{w})$)
  $u^\star \leftarrow u$
  $\tau^\star \leftarrow \tau^{(T)}(u^\star)$
  return $\tau^\star, u^\star$

V. INFERENCE

In this section, we describe how our planner obtains the desired behavior and trajectory. As shown in Algorithm 1 our inference process contains two stages of optimization. In the behavioral planning stage, we adopt a coarse-level parameterization for trajectory generation. The resulting trajectory is found by selecting the one with the lowest cost. In the trajectory planning stage, we use a fine-level parameterization where we model the trajectory as a function of vehicle control variables. The trajectory is initialized with the output of the behavior planner, and optimized through a continuous optimization solver.

A. Behavioral Planner

We represent a trajectory in terms of the Frenet Frame of the driving-path of candidate behaviors [2]. Let Γ_ρ be the transformation from a bicycle model state to the Frenet frame of a path ρ:

$$[s, \dot{s}, \ddot{s}, d, d', d''] = \Gamma_\rho(X), \tag{3}$$

where s is the position (arc length) along the path, d is the lateral offset. $\dot{(.)} := \frac{\partial}{\partial t}$, and $(.)' := \frac{\partial}{\partial s}$ denote the derivatives with respect to time and arc-length. Note that the longitudinal state is parametrized by time, but the lateral state is parametrized by the longitudinal position which, according

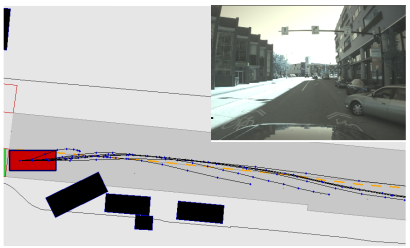

Fig. 4: Example trajectories in a nudging scenario.

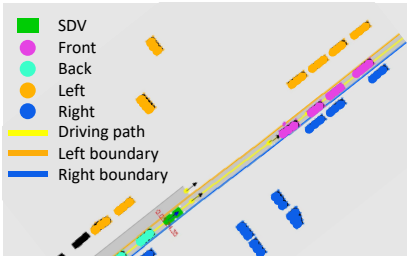

Fig. 5: Behavioral decisions include obstacle side assignment and lane information, which are sent through the behavioral-trajectory interface.

to [2], is a better representation of the coupling between the two states at relatively low speed. Figure 4 sketches the set of possible trajectories that can generate the nudging behavior of passing a vehicle. Given an initial vehicle state $\Gamma_{\rho_b}(X_0)$, we generate longitudinal and lateral trajectories as follows:

1) Logitudinal trajectories: The set of longitudinal trajectories $\mathcal{S} = \{s(t)\}$ are generated by computing an exhaustive set of mid-conditions $[\dot{s}(t_1), t_1]$ and end-conditions $[\dot{s}(T), T]$ and solving for two quartic polynomials stitched together. The acceleration ($\ddot{s}$) at t_1 and T are fixed at 0.

2) Lateral trajectories: Given a set of longitudinal trajectories, we parameterize lateral trajecotries $[d(s), d'(s), d''(s)]$ in terms of the longitudinal distance s. We generate a set of mid-conditions $[d(s_1), s_1]$ and fix $d'(s_1)$ and $d''(s_1)$ to be 0. We also fix the end-conditions to be $[0, 0, 0]$ so that the SDV is merged to the driving path. We stitch two quintic polynomials to fit the mid- and end-conditions.

For computing the dynamics cost in the discrete planner, we transform each pair of sampled longitudinal and lateral trajectories $[s(t), d(s)]$ back to a bicycle model trajectory:

$$\tau = [\mathbf{x}, \theta, \kappa, v, a, \dot{\kappa}] = \Gamma_{\rho}^{-1}(s, \dot{s}, \ddot{s}, d, d', d''). \quad (4)$$

Figure 4 shows an example set of generated bicycle model trajectories. The optimal trajectory for a given scenario $\mathcal{W}$ is found by evaluating the cost function f for all $b \in \mathcal{B}(\mathcal{W})$ and $\tau \in \mathcal{T}(b)$, and choosing the one that achieves the minimum cost.

B. Behavioral-Trajectory Interface

The behavioral-trajectory interface passes the optimal behavioral decision b^* and coarse-level trajectory τ^* to the trajectory planning stage. b^* is encoded as the left and right lane boundaries, the driving path, and the obstacle side assignment, which determines whether an obstacle stays in the front, back, left, or right to the SDV at time step t (see Fig. 5). For the trajectory planner, spatio-temporal overlap cost c_{overlap} will be incurred if the side assignment is violated at any time in the planning horizon, scaled by the squared distance of violation. This encourages that the trajectory planner respects the discrete decision made by the discrete stage.

C. Trajectory Planner

The behavioral planner uses finite differences to estimate the control parameters, which may not be precise for long range. Therefore, we use a trajectory fitter to compute the control parameters. In this stage, we represent trajectories in the Cartesian coordinates: $\tau = (\mathbf{x}, \theta, v, a, \kappa, \dot{\kappa})$. We parameterize the trajectory τ using the control variables jerk j and wrench $\ddot{\kappa}$ (second derivative of curvature). We use bicycle dynamics $\mathcal{D}$ to model τ as a function of the controls.

$$\tau_t = \mathcal{D}(\tau_{t-1}, u_t), \quad (5)$$

$$\tau = \{\mathcal{D}(\tau_{t-1}, u_t)\}_{t=1}^{T} = \tau^{(T)}(u). \quad (6)$$

Trajectory fitting minimizes the following objective wrt. the control variables:

$$\hat{u} = \underset{u}{\operatorname{argmin}} \sum_{t=1}^{T} c_{\mathbf{x}}(\mathbf{x}(u))_t + \lambda_\theta \sum_{t=1}^{T} c_\theta(u)_t + \lambda_{\text{dyn}} c_{\text{dyn}}, \quad (7)$$

where $c_{\mathbf{x}}$ is the squared Euclidean distance between the trajectory positions, c_θ is the orientation difference:

$$c_\theta(u)_t = \frac{1}{2} \left\| \begin{pmatrix} \sin\theta(u)_t \\ \cos\theta(u)_t \end{pmatrix} - \begin{pmatrix} \sin\hat{\theta}_t \\ \cos\hat{\theta}_t \end{pmatrix} \right\|_2^2, \quad (8)$$

and c_{dyn} is the set of costs related to vehicle dynamics described in Section IV-G. This allows us to start the optimization process with a physically feasible trajectory.

Given a fitted control sequence as initialization, the continuous optimization module achieves a local minimum of the overall cost function f:

$$u^\star = \underset{u}{\operatorname{argmin}}\, f(\tau^{(T)}(u), b^*, \mathcal{W}; \mathbf{w}). \quad (9)$$

We utilize the BFGS solver to obtain the solution of the above optimization problem.

VI. Learning

We use a combination of max-margin objective and imitation learning as our loss function

$$\mathcal{L}(\mathbf{w}) = \frac{\lambda_{\mathbf{w}}}{2} \|\mathbf{w}\|_2^2 + \lambda_M \mathcal{L}_M(\mathbf{w}) + \lambda_I \mathcal{L}_I(\mathbf{w}), \quad (10)$$

where $\mathcal{L}_M$ is the max-margin loss, $\mathcal{L}_I$ is the imitation learning loss, and λ_M and λ_I are hyperparameters that scale the two loss components.

Algorithm 2 Learning of our joint planner

1: **procedure** LEARNING($\mathbf{w}^{(0)}$)
2: **for** $i \leftarrow 1...N$ **do**
3: $E_M \leftarrow$ GETMINIBATCH, $\mathbf{g}_M \leftarrow 0$
4: $E_I \leftarrow$ GETMINIBATCH, $\mathbf{g}_I \leftarrow 0$
▷ Max-margin learning
5: **for** $E_j = (\tau_h, b_h, \mathcal{W}) \in E_M$ **do**
6: **for** $b \in \mathcal{B}, \tau \in \mathcal{T}(b)$ **do**
7: $l_D(b,\tau) \leftarrow \Delta(\tau_h, b_h, \tau, b) - f(\tau, b, \mathcal{W}; \mathbf{w}^{(i-1)})$
8: $\{b_k^*, \tau_k^*\} \leftarrow$ top-K $l_D(b,\tau)$ s.t. $b \in \mathcal{B}, \tau \in \mathcal{T}(b)$
9: $\mathbf{g}_M \leftarrow \mathbf{g}_M + \frac{1}{K|E_D|}\sum_k (\mathbf{c}_{\mathcal{W}}(\tau_h, b_h) - \mathbf{c}_{\mathcal{W}}(\tau_k^*, b_k^*))$
10:
▷ Differentiable inference imitation learning
11: **for** $E_j = (\tau_h, b_h, \mathcal{W}) \in E_I$ **do**
12: $u_0^\star \leftarrow$ OPTIMIZE($f(\tau^{(T)}(u), b^*, \mathcal{W}; \mathbf{w})$)
▷ Gradient descent M steps
13: **for** $m \leftarrow 1...M$ **do**
14: $u_m^\star \leftarrow u_{m-1}^\star - \eta \nabla_u f(u_m^\star)$
▷ Backprop through time (BPTT)
15: $l_C \leftarrow \frac{1}{T}\sum_t \gamma^t \|\mathbf{x}_t^\star - \mathbf{x}_{h,t}\|_2^2$
16: **for** $m \leftarrow M...1$ **do**
17: $\mathbf{g}_I \leftarrow \mathbf{g}_I + \frac{1}{|E_I|}\nabla_u l_C(u_m^\star)\nabla_{\mathbf{w}} u_m^\star$
18: $\mathbf{g} \leftarrow \lambda_{\mathbf{w}}\mathbf{w}^{(i-1)} + \lambda_M \mathbf{g}_M + \lambda_I \mathbf{g}_I$ ▷ Sum up gradients
19: $\mathbf{w}^{(i)} \leftarrow \mathbf{w}^{(i-1)} \exp(-\alpha \mathbf{g})$ ▷ Exp. gradient descent
20: **return** $\mathbf{w}^{(N)}$

The max-margin learning loss penalizes trajectories that have small cost and are different from the human driving trajectory [11]. Let $\{(\mathcal{W}, \tau_h, b_h)\}_{i=1}^N$ be a set of manual-driving examples where b_h and τ_h denote the ground-truth human behavior and trajectory respectively. We learn the linear weights $\mathbf{w}$ of the cost function f using structured SVM. This encourages the human driving trajectory to have smaller cost than other trajectories. In particular,

$$\mathcal{L}_M(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\Bigg\{ f(\tau_{h,i}, b_{h,i}, \mathcal{W}_i; \mathbf{w}) + \max_{b \in \mathcal{B}(\mathcal{W}_i), \tau \in \mathcal{T}(b)} \{\Delta(\tau_{h,i}, b_{h,i}, \tau, b) - f(\tau, b, \mathcal{W}_i; \mathbf{w})\} \Bigg\}, \quad (11)$$

and $\Delta(\tau_h, b_h, \tau, b)$ is the task-loss, which measures the dissimilarity between pairs of (τ, b). It consists of the L1 distance between the positions of the trajectories, and constant offsets for any behavioral differences and undesirable outcomes. The maximization can be solved by treating the task-loss as a sub-cost in f. In practice, we select the top K trajectories that have the maximal values.

For the imitation loss function $\mathcal{L}_I$, we use *mean square error* (MSE) for measuring the distance between positions of the human trajectory and the planner optimal trajectory, with some discounting factor γ:

$$\mathcal{L}_I(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\sum_{t=1}^{T}\gamma^t \|\mathbf{x}_t^\star - \mathbf{x}_{h,t}\|_2^2 \quad (12)$$

The overall gradient of the learning objective can be written as $\mathbf{g} = \lambda_{\mathbf{w}}\mathbf{w} + \lambda_M \mathbf{g}_M + \lambda_M \mathbf{g}_I$, where $\mathbf{g}_I = \nabla_{\mathbf{w}}\mathcal{L}_I$ and $\mathbf{g}_M$ is the subgradient of $\mathcal{L}_M$:

$$\mathbf{g}_M = \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\mathbf{c}(\tau_{h,i}, b_{h,i}, \mathcal{W}_i) - \mathbf{c}(\tau_{i,k}^*, b_{i,k}^*, \mathcal{W}_i), \quad (13)$$

where $\{\tau_k^*, b_k^*\}$ are K maximum violation examples and $\mathbf{c}$ is the sub-cost vector.

The max-margin objective uses a surrogate loss to learn the sub-cost weights, since selecting the optimal trajectory within a discrete set is not differentiable. In contrast, the iterative optimization in the trajectory planner is a differentiable module, where gradients of the imitation loss function can be computed using the backpropagation through time (BPTT) algorithm [23]. Since unrolling the full optimization can be computationally expensive, we unroll only for a truncated number of steps after we obtain a solution. As shown in Line 13 in Alg. 2, we perform M gradient descent steps after obtaining the optimal trajectory, and backpropagate through these M steps only (Line 16). If the control obtained from the continuous optimization converges to the optimum, then backpropagating through a truncated number of steps is approximating of the inverse Hessian ($[\nabla_{uu}^2 f]^{-1}$) at the optimum $u^\star$.

Since we would like the weights to be greater than zero, we use the exponentiated gradient descent update on the sub-cost weights: $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} \exp(-\alpha \mathbf{g})$, where α is the learning rate parameter. This update ensures that the learned weights are always positive.

VII. EXPERIMENTS

We conducted our experiments in *open-loop simulation*, where we unroll the planner independent of the new observations.

A. Datasets

We use two real-world driving datasets in our experiments. **ManualDrive** is a set of human driving recordings where the drivers are instructed to drive smoothly and carefully respecting all traffic rules. There are {12000, 1000, 1600} scenarios in the training, validation and test set respectively. We use this dataset for training and evaluation on major metrics. Since this dataset does not contain labeled objects, we additionally use **TOR-4D**, which is composed of very challenging scenarios. We exploit the labeled objects in 3D space to compute spatiotemporal overlap metrics. We use this dataset for evaluation only. In particular our test set contains 5500 scenarios.

B. Evaluation metrics

We use several metrics for evaluation.

a) Similarity to human driving: We use the average ℓ_2 distance between the planned trajectories and human trajectories within {1.0, 2.0, 3.0} seconds into the future. Lower number means that the planner is behaving similarly to a human driver.

b) Passenger comfort: We measure passenger comfort in terms of the average jerk and lateral acceleration.

Method	L_2 (m)			Driving Metrics				
	1.0s	2.0s	3.0s	Jerk	Lat. accel.	Speed lim.	Progress	Diff. behavior (%)
HUMAN	-	-	-	0.237	0.168	0.14	3.58	-
Oracle ACC	0.700	1.560	3.670	-	-	0.34	3.64	-
PT	0.254	0.941	2.190	0.147	**0.232**	**0.26**	**3.81**	8.5
Ours $\mathbf{w}$	0.095	0.712	1.992	0.140	0.264	0.39	**3.81**	**0.0**
Ours $\mathbf{w}_b$	0.094	0.699	1.945	**0.135**	0.234	0.27	3.72	**0.0**
Ours $\mathbf{w}_b^t$	**0.066**	**0.514**	**1.495**	0.143	0.279	0.28	3.78	**0.0**

TABLE I: Human L_2 and driving metrics on ManualDrive

Method	All			excl. non-overlap pred.		
	1.0s	2.0s	3.0s	1.0s	2.0s	3.0s
Oracal ACC	0.780	1.300	2.990	-	-	-
PT	0.290	1.480	3.230	0.036	0.670	2.130
Ours $\mathbf{w}$	**0.000**	**0.128**	1.010	**0.000**	0.055	0.600
Ours $\mathbf{w}_b$	**0.000**	**0.128**	1.010	**0.000**	0.055	0.560
Ours $\mathbf{w}_b^t$	**0.000**	**0.128**	**0.826**	**0.000**	**0.036**	**0.340**

TABLE II: Spatiotemporal overlapping rate (%) on TOR-4D

c) Spatiotemporal overlap: A good planning should avoid obstacles. We measure the spatiotemporal overlap with obstacles within $\{1.0, 2.0, 3.0\}$ seconds into the future, in terms of the percentage of scenarios. We use a perception and prediction (P&P) module as input to the planner. We also report the percentage of overlap excluding the obstacles that are not present in detection or predicted not to have overlap with the SDV, including vehicles that are more than 25m behind the SDV as they are assumed to react to the ego vehicle (referred to as "excl. non-overlap pred.").

d) Others: We also measure other aspects of driving: average violation of speed limit (km/h), progress in the route, and proportion of the scenarios where the planner chooses a different behavior compared to the human.

C. Baselines

We use **Oracle ACC** as baseline. Note that this baseline uses the ground-truth driving path, and follows the leading vehicle's driving trajectory. We call this oracle as it has access to the ground-truth behavior. Our second baseline, called **PT**, is a simpler version of our behavioral planner. It only reasons in the longitudinal-time space, without the lateral dimension. In lane change decisions, it projects obstacles from both lanes onto the same longitudinal path. The weights are also learned using the max-margin formulation described in Section VI

D. Experiment setup

As described in IV, we have 30 different sub-costs. We consider the following **model variants** with increasing complexity:

- $\mathbf{w}$ learns a single weight vector for all scenarios. This model learns 30 different weights.
- $\mathbf{w}_b$ learns a separate weight vector for each behavior: keep lane, left lane change, and right lane change. Thus it has $3 \times 30 = 90$ weights for 3 behaviors.
- $\mathbf{w}_b^t$ learns a separate weight vector for each behavior and at each time step. This variant can automatically learn planning cost discounting factor for the future. This model has $3 \times 30 \times 21 = 1890$ weights for 21 timesteps.

The following **ablation variants** are designed to validate the usefulness of our proposed joint inference and learning procedure.

- *Behavioral with max-margin ("B+M")* learns the weight vector through the max-margin (+M) learning on the behavioral planner only.
- *Full Inference ("B+M +J")* uses the trained weights of "B+M", and runs the joint inference algorithm (+J) at test time.
- *Full Learning & Inference ("B+M +J +I")* learns the weight vector using the combination of max-margin (+M) and imitation objective (+I), and runs the joint inference algorithm (+J) at test time.

Training details: We pretrain the behavioral planner with the max-margin objective for a fixed number of steps, and then start joint training of the full loss function. For fair comparison, the baseline (B+M) is also trained for the same number of steps. For learning the imitation loss during training, we use human trajectory controls as the initialization for the trajectory optimization for training stability.

E. Results and Discussion

Table I presents our results on the ManualDrive dataset. The best model $\mathbf{w}_b^t$ clearly outperforms other variants and baselines. In terms of similarity to human trajectories within 3.0s time horizon, the full model has a relative improvement of 31.7% over the PT baseline which uses a simpler cost function without a trajectory planner, and 24.9% over the simplest model variant $\mathbf{w}$. Although the full model is still a linear combination of all the sub-costs, $\mathbf{w}_b^t$ has almost 2000 free parameters and it would be virtually impossible to manually tune the coefficients. Our jointly learned models also shows better behavioral decision compared to PT. We also provide other driving metrics in the table; however, they are less indicative of the model performance. For example, the human driving trajectories are reported to have a higher jerk and make less distance progress.

Table II presents our results on the TOR-4D dataset, where we measure the spatiotemporal overlapping of our SDV with other obstacles. Our jointly learned model shows lower chance of overlapping with obstacles: compared to the PT baselines, our best model reduced 73% of the spatiotemporal overlapping within 3.0s time horizon.

Joint inference and learning: Table III,IV presents results when the joint inference or learning modules are removed from our system. Shown in Table III, the full model shows a

Method	L_2 (m)			Driving Metrics			
	1.0s	2.0s	3.0s	Jerk	Lat. accel.	Speed lim.	Progress
$\mathbf{w}$ B+M	0.241	0.900	2.100	0.143	0.257	**0.22**	3.72
$\mathbf{w}$ B+M +J	0.096	0.727	2.030	0.141	**0.239**	0.39	3.80
$\mathbf{w}$ B+M +J +I	**0.095**	**0.712**	**1.992**	**0.140**	0.264	0.39	**3.81**
$\mathbf{w}_b$ B+M	0.241	0.890	2.070	0.145	0.282	**0.23**	3.73
$\mathbf{w}_b$ B+M +J	0.097	0.720	2.000	0.143	0.264	0.37	**3.79**
$\mathbf{w}_b$ B+M +J +I	**0.094**	**0.699**	**1.945**	**0.135**	**0.234**	0.27	3.72
$\mathbf{w}_b^t$ B+M	0.240	0.790	1.750	**0.136**	0.330	**0.20**	3.72
$\mathbf{w}_b^t$ B+M +J	**0.066**	**0.514**	1.501	0.143	**0.278**	0.29	**3.80**
$\mathbf{w}_b^t$ B+M +J +I	**0.066**	**0.514**	**1.495**	0.143	0.279	0.28	3.78

TABLE III: Effect of joint learning and inference on ManualDrive

Method	All			excl. non-overlap pred.		
	1.0s	2.0s	3.0s	1.0s	2.0s	3.0s
$\mathbf{w}$ B+M	0.110	0.330	1.390	0.018	0.091	0.690
$\mathbf{w}$ B+M +J	**0.000**	**0.110**	1.020	**0.000**	**0.055**	0.620
$\mathbf{w}$ B+M +J +I	**0.000**	0.128	**1.010**	**0.000**	**0.055**	**0.600**
$\mathbf{w}_b$ B+M	0.011	0.330	1.370	0.018	0.091	0.690
$\mathbf{w}_b$ B+M +J	**0.000**	**0.128**	1.040	**0.000**	**0.055**	0.580
$\mathbf{w}_b$ B+M +J +I	**0.000**	**0.128**	**1.010**	**0.000**	**0.055**	**0.560**
$\mathbf{w}_b^t$ B+M	0.110	0.360	1.130	0.018	0.091	0.360
$\mathbf{w}_b^t$ B+M +J	**0.000**	0.146	0.936	**0.000**	**0.018**	0.360
$\mathbf{w}_b^t$ B+M +J +I	**0.000**	**0.128**	**0.826**	**0.000**	0.036	**0.340**

TABLE IV: Spatiotemporal overlapping rate (%) on TOR-4D

clear improvement in terms of the ℓ_2 position error compared to human. This is expected since it is optimized as the imitation loss. Compared to the model that does trajectory planner only for inference, the jointly learned model shows better performance in terms of jerk and lateral acceleration. In Table IV, the full model achieves the lowest spatiotemporal overlap rate. This suggests that by treating the trajectory optimization as a learnable module, our planner learns to produce safe and smooth trajectories by imitating human demonstrations.

VIII. Conclusion

In this paper we proposed a learnable end-to-end behavior and trajectory motion planner. Unlike most existing learnable motion planners that address either behavior planning, or use an uninterpretable neural network to represent the entire logic from sensors to driving commands, our approach features an interpretable cost function and a joint learning algorithm that learns a shared cost function employed by our behavior and trajectory components. Our experiments on real self-driving datasets demonstrate that the jointly learned planner performs significantly better in terms of both similarity to human driving and other safety metrics. In the future, we plan to explore utilizing our jointly learnable motion planner to train perception and prediction modules. This is possible as we can back-propagate through it. We expect this to significantly improve these tasks.

References

[1] Zlatan Ajanovic, Bakir Lacevic, Barys Shyrokau, Michael Stolz, and Martin Horn. Search-based optimal motion planning for automated driving. In *IROS*, 2018.

[2] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *ICRA*, 2010.

[3] Philipp Bender, Omer Sahin Tas, Julius Ziegler, and Christoph Stiller. The combinatorial aspect of motion planning: Maneuver variants in structured environments. In *IV*, 2015.

[4] Tianyu Gu, John M Dolan, and Jin-Woo Lee. On-road trajectory planning for general autonomous driving with enhanced tunability. In *Intelligent Autonomous Systems 13*, pages 247–261. Springer, 2016.

[5] Haoyang Fan, Fan Zhu, Changchun Liu, Liangliang Zhang, Li Zhuang, Dong Li, Weicheng Zhu, Jiangtao Hu, Hongye Li, and Qi Kong. Baidu apollo em motion planner. *arXiv preprint arXiv:1807.08048*, 2018.

[6] Julius Ziegler and Christoph Stiller. Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios. In *IROS*, 2009.

[7] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *J. Field Robot.*, 26(3):308–333, March 2009.

[8] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *ICRA*, pages 4889–4895, 2011.

[9] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. Trajectory planning for berthaa local, continuous method. In *IV*, 2014.

[10] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[11] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *ICML*, 2006.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[13] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[14] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018.

[15] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[16] Haoyang Fan, Zhongpu Xia, Changchun Liu, Yaqin Chen, and Qi Kong. An auto-tuning framework for autonomous vehicles. *arXiv preprint arXiv:1808.04913*, 2018.

[17] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[18] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining neural networks and tree search for task and motion planning in challenging environments. In *IROS*, 2017.

[19] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *arXiv preprint arXiv:1807.00412*, 2018.

[20] Nicholas Rhinehart, Kris M Kitani, and Paul Vernaza. R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *ECCV*, 2018.

[21] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *CVPR*, 2019.

[22] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.

[23] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.