Imitative Planning using Conditional Normalizing Flow

Shubhankar Agarwal[1]*, Harshit Sikchi[2]*, Cole Gulino*, Eric Wilkinson* and Shivam Gautam[3]*

Abstract—**A popular way to plan trajectories in dynamic urban scenarios for Autonomous Vehicles is to rely on explicitly specified and hand crafted cost functions, coupled with random sampling in the trajectory space to find the minimum cost trajectory. Such methods require a high number of samples to find a low-cost trajectory and might end up with a highly suboptimal trajectory given the planning time budget. We explore the application of normalizing flows for improving the performance of trajectory planning for autonomous vehicles (AVs). Our key insight is to learn a sampling policy in a low-dimensional latent space of expert-like trajectories, out of which the best sample is selected for execution. By modeling the trajectory planner's cost manifold as an energy function, we learn a scene conditioned mapping from the prior to a Boltzmann distribution over the AV control space. Finally, we demonstrate the effectiveness of our approach on real-world datasets over IL and hand-constructed trajectory sampling techniques.**

I. Introduction

Generating a control trajectory which provides safe, comfortable, and socially responsible motion is a fundamental problem for operating autonomous vehicles (AVs). Since high quality human driving data is easily available, imitative models which learn to mimic expert demonstrations are a popular approach [1]. End-to-end imitation learning (IL) approaches are attractive because they allow for a mapping to be learned between high dimensional context features, such as sensor and map data, and the control space of the vehicle platform.

However, these IL approaches have several limitations which make their use in practice difficult. The first is that for every scene there is only one label, since the expert only provided one demonstration, and it is unclear how to properly penalize deviations from the demonstration. This is the commonly known distribution shift problem [2], and a lack of an accurate simulator precludes us from correcting the distribution shift. The second is that the internal belief state of the expert is not available which means the AV is unlikely to learn the correct response to it's own aleatoric and epistemic uncertainties about a road scene. Finally, AV operation typically requires high confidence in the safety outcomes of a control trajectory, which typically necessitates a whitebox costing module to certify the IL method's output.

In this work, we propose a method to address these problem by treating the whitebox costing module as an energy based model and learning a sampling policy that minimizes a certain f-divergence to it. Furthermore, we restrict the policy actions to a lower dimensional latent space, which is trained to encode trajectories obtained from the expert demonstrations. Whitebox planners ingest interpretable representations of the scene, which enables the enforcement of strong conditions on safety, and can reason about the uncertainties of the AV system. Additionally, in contrast to a single expert demonstration, the cost manifold provides information about how to penalize deviations from the optima. Thus, instead of learning the PDF of the expert given a scene the proposed method learns a density function which corresponds to the planner's cost manifold. Specifically, we used normalizing flows to learn the density function, because of their ability to learn complicated multimodal distributions.

Our approach builds upon normalizing flows which are capable of representing complex, multimodal manifolds from a known prior distribution and supports efficient, parallel sampling. First, we use a variational autoencoder to learn a representative subspace of the control trajectory space from all expert driving demonstrations. Samples from this encoding space generate control trajectories which behave *stylistically* the same as the expert, or encode trajectories that are kinematically similar to expert. Then we learn a normalizing flow mapping from the prior distribution to a Boltzmann distribution in the control trajectory encoding space using the cost manifold as an energy function. We propose using neural autoregressive flow (NAF) [3] for this flow mapping because of it's ability to learn complicated multimodal target distribution, while performing accurate PDF inference. We train our method following the inverse autoregressive flow (IAF) [4] which allows for efficient control trajectory sample generation using parallel transformations. Subsequently, we will refer to our method as *FlowPlan*.

The main contributions of this work are:

- An efficient method to generate trajectories for autonomous driving by learning a sampling policy in a scene-conditioned low-dimensional latent space representative of expert driving demonstrations.
- We demonstrate the utility of normalizing flow by taking advantage of the exact pdf inference to further refine our generated trajectories without the whitebox costing module. As a by-product of our sampling policy we can efficiently generate scores (log probs) of the sampled trajectories without the whitebox costing module.
- We demonstrate the benefits of our approach over hand constructed, parametric sampling strategies on real world datasets.

*Work done while at Aurora Innovation.

[1]Department of Electrical and Computer Engineering, The University of Texas at Austin, [2]Department of Computer Science, The University of Texas at Austin, [3] Aurora Innovation {somi.agarwal, hsikchi, }@utexas.edu

II. Related Work

Trajectory sampling techniques for planning attempt to construct trajectories from structured, parametric representations which are likely to solve the SDV's planning problem. One common method used for in-lane driving is to construct samples within a Frenet frame around a nominal path as explored by [5] with traffic-adaptive velocity profiles for highway driving. A review encompassing these approaches including clothoid, bezier, and polynomial representations can be found in [6]. In contrast to our approach, these methods typically involve hand crafting strategies for adapting the parameters of the trajectory representation to the planning problem.

Variational methods which perform continuous optimization in a function space are typically solved with iterative strategies such as DDP [7] or iLQR [8]. These methods can only represent a small subset of the real world problems, i.e. convex problems or quadratic in case of iLQR, while most of the self-driving problems are non-convex. A survey of this class of approaches can be found in [9]. Our work complements these methods since we present a data-driven framework for planning without placing any assumptions on convexity of the problem.

Imitation Learning Methods Learning-based approaches have recently gained momentum in generating motion trajectories, Imitation Learning (IL) being one of the most popular approach. In IL, expert demonstrations are used to learn the desired behavior or driving policy, [10] being one of the first successful demonstration. Since then, significant progress has been made to accomplish more complex maneuvers and scenarios, in [11], [12]. But, these approaches are not able to generalize outside the expert demonstrations as shown in [13]. [1] and [14], address generalization outside expert demonstrations by doing closed-loop training and adding different goal functions to guide imitation policy respectively. While these IL approaches alleviate the need for hand-tuning cost functions, they suffer from compounding errors due to autoregressive nature and provide very little or no interpretability.Prior work has also explored using normalizing flows for learning a density function over trajectories. [15] combine a conditional normalizing flow model with VAE to learn an invertible density model for trajectory sampling from expert demonstration. [16] investigate conditional VAEs with endpoint conditioning to accomplish goal-directed sampling along with a social pooling layer for capturing interaction. Our work is distinct from these are we leverage a whitebox cost function to generate reliable and compliant trajectories while using expert driving behaviors as composable *skills*.

Inverse Reinforcement Learning Inverse Reinforcement Learning (IRL) based approaches have been used to learn the motion planning cost functions (alternatively reward functions), removing the need for hand constructing cost functions for autonomous driving. IRL's early application in autonomous driving stems from [17]. Since then, IRL has been used for motion planning in [18], [19] and [20]. Most of these approaches use IRL to make discrete decisions (pass, yield, etc.) and operate in very specific simulated scenarios with significantly smaller feature space than the real world. IRL methods like [21], [22] require access to a simulator for extracting the cost functions. Since it is infeasible to let the AV explore in the real world, the use of a simulator is required, and simulator inaccuracy can further lead to sim-to-real transfer issues.

III. Background

A. Motion Planning Problem

The purpose of the planner is to provide safe, comfortable motion for an autonomous vehicle constrained by dynamic and kinematic feasibility, partial observability, and user experience preferences. This is accomplished by formulating the problem as a partially observable Markov decision process (POMDP) which is optimized over a finite time horizon T. In this work the POMDP model is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{Z}, \mathcal{C}, b \rangle$ where $\mathcal{S}$ is the state space of the scene, $\mathcal{A}$ is the action space, $\mathcal{O}$ is the observation space, and $\mathcal{T}(s'|s,a)$ is the probabilistic transition function from state s to s' when taking action a. The belief state $b_t(s)$ is a probability distribution over the scene states $s \in \mathcal{S}$ which the AV maintains from the history of observations and actions $h_t = (o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1})$ and the initial belief state b_0. The observation and transition models allow for the belief state to be updated through Bayes rule. $\mathcal{Z}$ is the partition function. $\mathcal{C}$ is the cost function, specifically for taking an action a at state s given belief state b under policy π. A complete description of POMDPs can be found in [23].

In this work the policy π is a stochastic mapping $\mathcal{B} \rightarrow \mathcal{A}^T$ from belief space to the action sequence of horizon T. We formulate the planner cost as an energy based model [24] which define a Boltzmann distribution using exponentiated cost functions i.e $\pi(\mathbf{a}|b) \propto \prod_{t=0}^{T-2} e^{-\mathcal{C}(s_t,a_t|\pi,b_t)} \cdot e^{-\mathcal{C}_T(s_{T-1},a_{T-1}|\pi,b_T)}$ where $\mathbf{a}$ is the action sequence and $\mathcal{C}_T$ is a terminal cost function that approximates the remaining cost-to-go. The performance of the policy is given by:

$$J(\pi|b) = \mathbb{E}_{s_0 \sim b, s_t \sim \mathcal{T}, \mathbf{a} \sim \pi(.|b)} \tag{1a}$$

$$\left[\frac{1}{\mathcal{Z}} \prod_{t=0}^{T-2} e^{-\mathcal{C}(s_t,a_t|\pi,b_t)} \cdot e^{-\mathcal{C}_T(s_{T-1},a_{T-1}|\pi,b_T)} \right] \tag{1b}$$

The planner performs an online search for the optimal deterministic policy π^* which maximizes the expected value of the distribution under the belief state b

$$\pi^* = \underset{\pi \in \mathcal{P}}{\arg\max}\; J(\pi|b) \tag{2}$$

B. Normalizing Flows

A finite normalizing flow (flow) is an iterative framework for estimating and building flexible target distributions introduced in [27]. The flow model consists of a series of invertible transformations τ_n which map a known prior distribution $q(z_0)$ to a potentially complex, target distribution while preserving the total probability mass of the original *pdf*. More formally,

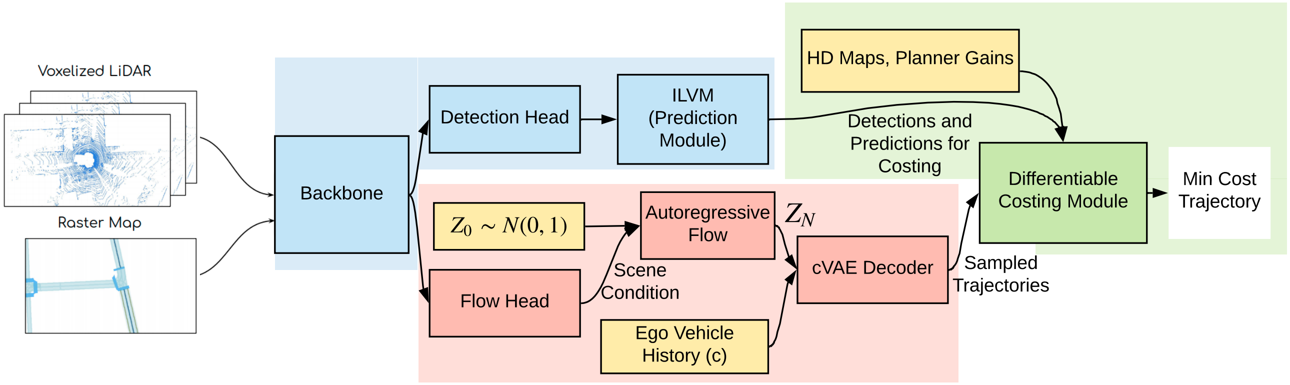


Fig. 1. Our model architecture for FlowPlan. Blue modules are used from previous work [25], [26] which are pretrained and are kept frozen during training. Red modules represent our flow planner which works as a control trajectory sampling module. Green modules represent the components of a traditional trajectory planner.

$$z_0 \sim q(z_0) \tag{3}$$

$$z_N = \tau_n(z_{n-1}; \theta | h_t), \quad \forall n = 1....N \tag{4}$$

where θ are the parameters of the flow model transformations and are conditioned on the action and observation history h_t. Since each transformation is invertible, we can use the change of variables formula to obtain the final log density:

$$\log q(z_N|h_t) = \log q(z_0|h_t) - \sum_{n=1}^{N} \log \det \left| \frac{dz_n}{dz_{n-1}} \right| \tag{5}$$

We can think of transformations τ_n as expanding or contracting the space of the known prior $q(z_0)$ into the conditional target $q(z_N|h_t)$ with the corresponding Jacobian determinant describing the relative change of volume and ensuring total probability mass is conserved.

IV. Method

An overview of our model architecture for FlowPlan can be found in Figure 1. Raw sensor data (LiDAR, Cameras, Radars) and HD map data is processed by a backbone network, to construct an internal feature representation. Actor detections and future predictions are generated from the output of backbone network using separate deep networks, described in section IV-A. The detections, predictions and HD maps are used by the whitebox costing module to provide the cost for each trajectory and is described in section IV-B. A σVAE is used to learn a reduced dimensional latent space of the trajectory control samples from expert human demonstrations, described in section IV-D. Our flow network works in parallel to the detector head and also consumes the output of backbone network as a scene conditioner. An autoregressive flow conditioned on the scene, generates trajectory samples in a latent space, described in section IV-C. The flow module is trained to minimize the loss function defined in section IV-C.

A. Scene Conditioning

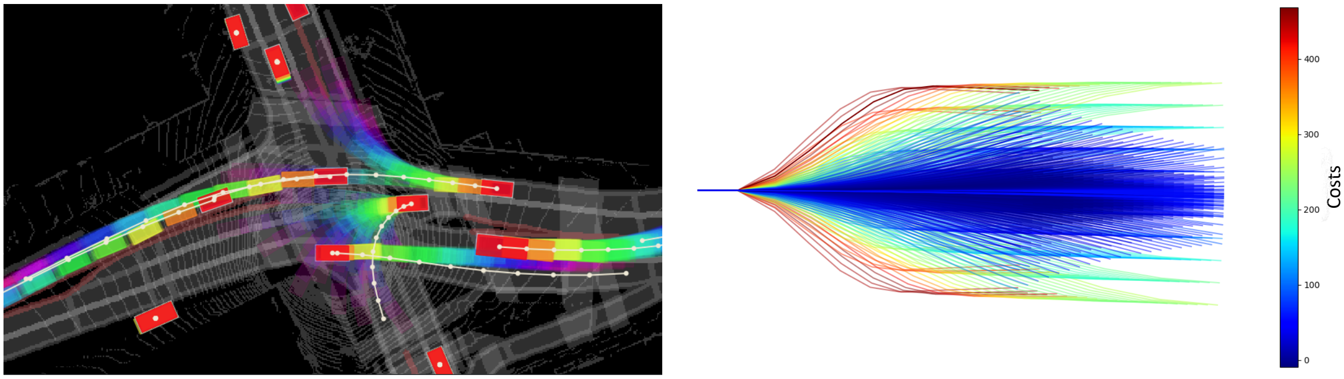
An AV's observation and action history h_t is high dimensional, consisting of a historical sequence of sensor observations, map states, and vehicle states. Starting from this raw data, we seek to construct a context feature vector representing the belief state $b(h_t)$ for conditioning the flow network. In this work, we use a pretrained detector [25] which takes as input a voxelized LiDAR point cloud and rasterized map state and constructs an internal feature representation of $b(h_t)$, which we denote as $\bar{b}(h_t)$. Output from this detector head is consumed by the prediction head (ILVM [26]) for generating scene predictions for the future. ILVM is a graph neural network used for generating multimodal future actor distributions. Actors are agents in the environment external to our AV, such as pedestrians and other vehicles. These actor trajectory predictions are passed to the whitebox costing module and utilized to give an interpretable scalar cost as an output. Simultaneously, the belief state $\bar{b}(h_t)$ output by the detector head is consumed as a scene conditioner for the flow-plan module. An illustration of the ILVM output can be seen in Figure 2(a).

B. Trajectory Planner

The purpose of a trajectory planner in an AV system is to find control policy π^* corresponding to the optima of the cost manifold from Eq. 2. In this work, the planner comprises of two parts: a control-trajectory sampling scheme and an interpretable whitebox costing module. The output of the planner is a deterministic control trajectory which provides minimum expected cost. The sampling scheme is our main contribution and discussed in section IV-C.

C. Sampling Policy

The whitebox costing module is a linear combination of a number of cost functions, encoding preferences for safety, per-



(a)

(b)

Fig. 2. **A self-driving scenario with actor predictions (a):** The predicted trajectories for all the actors in the scene obtained via the ILVM [26] network. The color gradient shows different timesteps in the predicted trajectory. White curves illustrate the ground truth behavior of the actor. **Output of the baseline Polynomial Frenet method (b):** Control trajectories in continuous (x,y) frame generated from the baseline Polynomial Frenet method over a straight path. The color gradient from blue to red indicates the change in costs for the control trajectories considering dynamics and lane following penalties.

formance and user comfort. We provide detailed descriptions of the cost functions in Appendix F. We utilize the costing module in two ways: During offline training as supervision to learn a stochastic sampling policy, and in online testing to select the best trajectories for execution. The module ingests map data, vehicle platform state, and probabilistic multimodal trajectory predictions for other actors future states to generate a scalar expected cost. We rely on Dubins model to simulate forward dynamics of the AV. A key requirement is that the cost functions and dynamics used for forward propagation to be differentiable to support training. We implement a differentiable forward simulator using 2-D bicycle dynamics [28] represented as a deferentially flat system [29].

D. Expert Demonstration Encoding

We construct an encoding space of human expert control trajectory demonstrations using a conditional Variational Autoencoder (cVAE). The cVAE learns a lower dimensional subspace of human-like trajectories using a large dataset of human demonstrations. The inputs to the VAE are x which is the human expert control trajectory and a condition vector c which consists of a fixed-length history of AV control trajectory. The cVAE is trained following the σVAE [30] method which allows for the weight between the MSE and KL divergence terms in the loss function to be learned removing the need of additional hyperparameter tuning.

$$\mathcal{L} = D \ln \sigma + \frac{D}{2\sigma} MSE(\hat{x}, x) + D_{KL}(q(z|x)||p(z)). \quad (6)$$

where the first two terms are reconstruction error under a gaussian decoder parameterization and the last term controls distance to the prior (similar to [30]).The σVAE is pre-trained on all human expert control trajectories from the dataset and is kept frozen during the flow training. During the main training loop only the decoder is used to decode trajectory samples from the latent space.

We propose to learn a stochastic sampling policy in the latent space of the cVAE described above. Towards this goal, We formulate the cost functions as an energy based model (Eq.1) and learn a maximum entropy policy minimizing the reverse KL divergence to it. We utilize Neural Autoregressive Flow (NAF) to facilitate efficient learning of multimodal energy landscape induced by the cost as well as to obtain concrete probability estimates which provides a score for each sampled trajectory. The affine transformations which were used in earlier flow models such IAF [4] and MAF [31] supported efficient inversion and log determinant calculation required for (5) but are not as flexible in representing multimodal distribution as NAF as shown in [3]. Our NAF policy will take as input a vector sampled from a prior distribution ($z_0 \sim \mathcal{N}(0, 1)$) and the belief state of the AV ($\bar{b}(h_t)$) as a conditioner. It outputs z_N in the latent space of our VAE, as a result of a number of flow transformations along with its probability.

In this work, we aim to learn a mapping from a known prior distribution, $q(z_0) = \mathcal{N}(0, 1)$, to the target distribution defined by the planner cost surface in Eq. 1. We formulate the mapping as the optimization as a reverse-KL divergence minimization:

$$\text{argmin}_\theta D_{\text{KL}} \left[q(z_N|\theta, b) \;||\; J(z_N|b) \right] \quad (7)$$

where $q(z_n|\theta, b)$ is the output of the flow model in Eq. 5 and $J(z_n|b)$ is the likelihood of that output under the planner cost surface. We train the normalizing flow policy by obtaining the scene context feature vector from the backbone network as described in section IV-A and drawing L samples from the prior distribution $z_0 \sim \mathcal{N}(0, 1)$. The per sample loss for reverse KL (Equation 7) can be simplified as follows:

$$L(b) = -\log(J(z_N|b)) - \sum_{n=1}^{N} \log \det \left| \frac{dz_n}{dz_{n-1}} \right| \quad (8)$$

The above loss function uses the cost function J to learn a generator parameterized as a normalizing flow with prior z_0. Here, we can ignore the partition function as it does

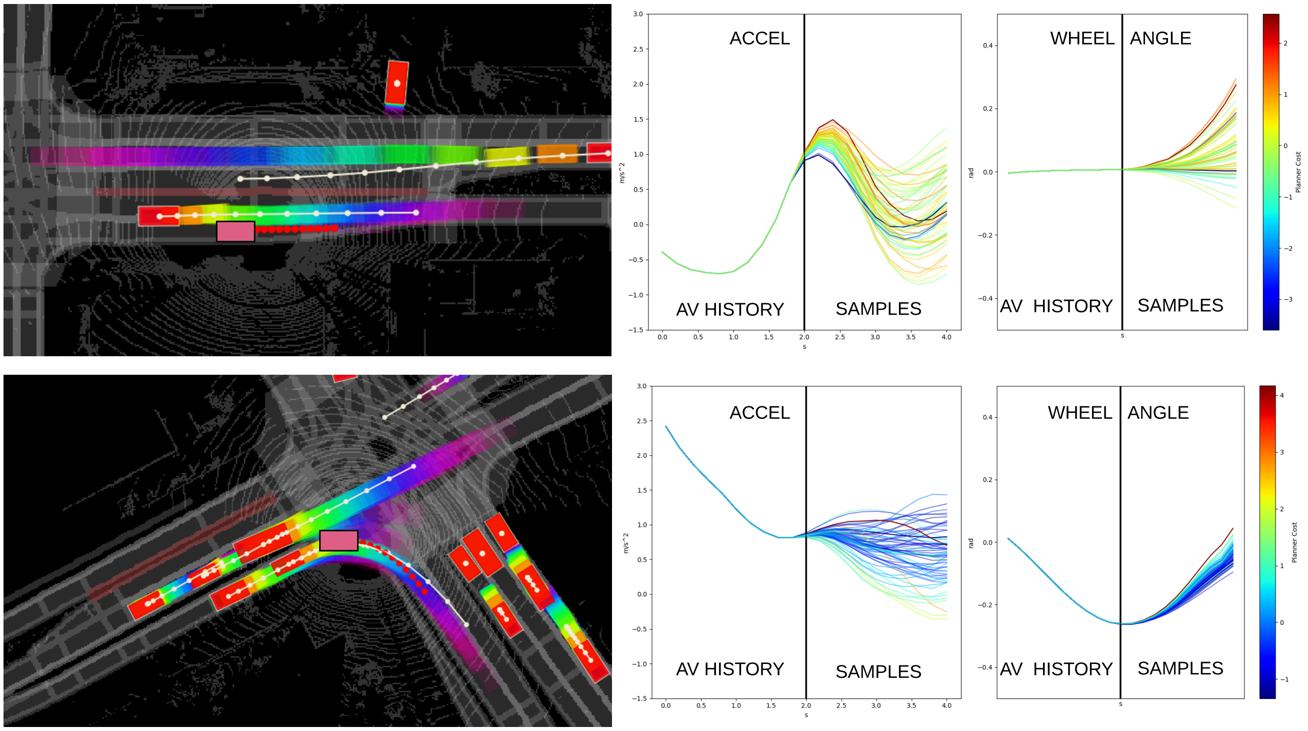


Fig. 3. **Outputs of the FlowPlan on two challenging self-driving scenarios.** In the left image, the AV is a pink box, and red dots represent the chosen trajectory under the planner cost. **Top Left:** A scenario where the AV must wait to merge behind an incoming actor which has priority. Our model generates a variety of control trajectories that decelerate and preemptively steer for lane alignment. **Bottom Left:** A scenario where the AV is making a right turn while staying in the lane, showing the importance of considering lane boundaries through the planner cost surface. **Right Images:** 64 sampled trajectories from the FlowPlan model for the respective scene. The control trajectories' color represents the trajectory's respective cost under the planner cost surface. The area left of the black line in the samples plot indicates AV's 2 sec controls history. In the top scenario, trajectories merging the lane have high costs, while trajectories staying in the lane have lower costs as expected (can be observed in the wheel angle image). In the bottom scenario, the trajectories making the right turn have lower costs, and the other trajectories going out of the lane have higher costs can be observed in the wheel angle image). **The key takeaway is that our FlowPlan can operate in challenging scenarios with minimal sampled trajectories compared to the *Polynominal Frenet* method.**

not depend on the parameter θ. We elaborate more on the partition function in Appendix F. The obtained solution can be interpreted as a maximum entropy sampling distribution for the whitebox planner. A maximum entropy policy can be proved to be a solution of robust-reward control problem in the presence of an adversary as shown in [32]. Even in the setting without an adversary, the adversarial objective bounds the worst case performance of the agent. This is similar to the policies obtained from state of the art model-free Reinforcement Learning method SAC [33]. The solution to the planning problem (Eq.2) is given by the maximum aposteriori estimate (MAP) under the learned policy parameterized by θ.

V. Experimental Results

Our proposed method and ablations are compared on the autonomous driving dataset HES-4D [34]. HES-4D uses a 64-beam roof-mounted LiDAR and consists of 6500 snippets in total, each 25 seconds long, spanning multiple North American cities. In each city, we have access to high definition maps capturing the geometry and the topology of each road network. The perceptual RoI including sensor and map data is 140×80 meters centered on the self-driving vehicle and for contextual history we utilize a total of 10 LiDAR sweeps at 10 Hz. The pretrained perception and prediction model (Section IV-A) predicts 2 seconds into the future at 0.2 seconds intervals. All models in this section utilize the pretrained Pixor [25] + ILVM [26] network for generating the motion forecasts for other actors used in the whitebox planner costs functions as well as for generating the perception backbone feature vector $\bar{b}(h_t)$.

The σVAE (Section IV-D) is pretrained using the human driving examples from the HES-4D and is conditioned with the AV history consisting of 2 seconds of dynamics information at 5 Hz. Our costing module design uses the cost functions as described in [35], which consists of number of costs including cost for vehicle dynamics (accel, jerk, streering angle etc.), lane violations, collision penalty, distance travelled along path among others. All samples generated from prior distributions come from $\mathcal{N}(0, 1)$ unless otherwise specified. For a baseline method we use a popular Frenet frame method similar to [5] and generate trajectory samples from the cross product of independent polynomials in the longitudinal

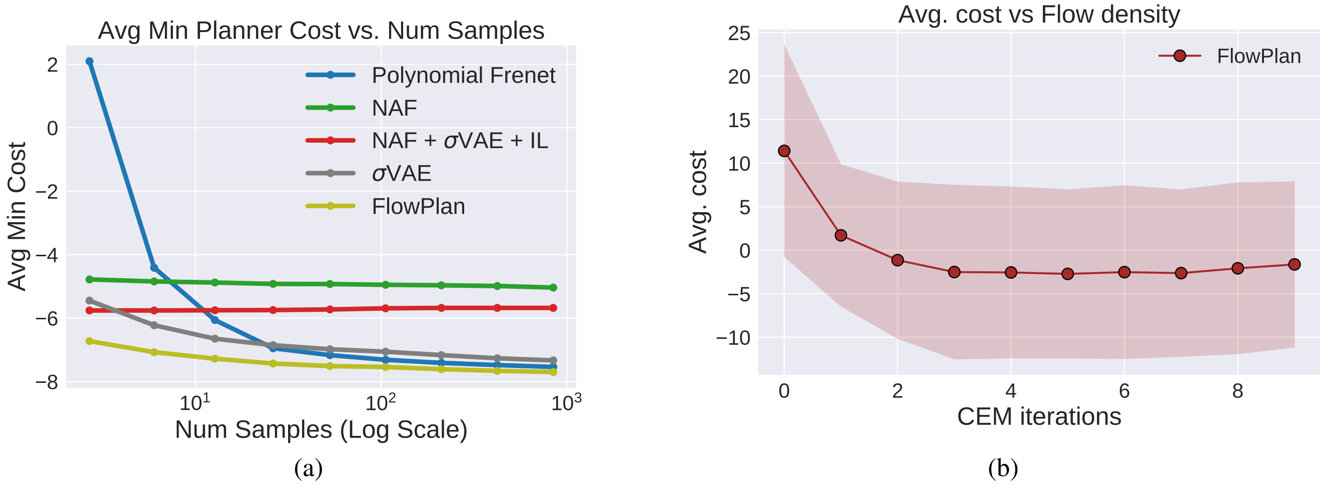


Fig. 4. **Sampling efficiency of the FlowPlan compares to the baselines (a):** A comparison of different sampling techniques used for generating low-cost control trajectories. We measure the average cost of the best-performing control trajectory for every scene in the evaluation set. Our method, FlowPlan, outperforms the baseline Polynominal Frenet method, especially in the low sample regime. **High-probability regions in the learned distribution map to lower-cost trajectories (b):** We demonstrate that high probability regions of the NAF output distribution correspond to low-cost surface regions using the cross-entropy method (CEM). As CEM iterations increase, the corresponding average cost under the planner cost surface of all samples in the CEM set decreases.

and lateral dimensions. The polynomials are generated using uniform distribution of end point conditions usually specified in terms of lateral and longitudinal displacement, end-point velocities and accelerations. This is referred to in the results as *Polynominal Frenet* method. Figure 2(b) shows example control trajectories generated by *Polynominal Frenet*. Control trajectory samples for all methods consist of acceleration and steering angle tuples for 2 seconds futures at 5 Hz. We provide detailed descriptions of our dataset and model architectures in Appendix A.

We provide qualitative results in Fig. 3 for our model. These results demonstrate the effectiveness of the model and the importance of learning the planning cost surface during training.

A. *Sampling Efficiency*

We evaluate the sampling efficiency of various approaches by measuring the average planner cost of the best performing sample across the evaluation dataset as a function of number of control trajectories generated. Lower cost implies that the sampled control trajectory is closer to the optima of the cost surface. In Figure 4 (a), we compare our method FlowPlan to a number of baselines- 1. Polynomial Frenet method, and a number of architectural variations of FlowPlan for learning the sampling policy, 2. NAF: Sampling policy is learned in complete trajectory space, 3. σVAE: A σVAE trained on human demonstrations is directly queried for the future trajectories given the history, 4. Context σVAE: A σVAE trained on human demonstrations is directly queried for the future trajectories given the history and the scene context, 5: NAF+σVAE+IL: Uses the same architecture as FlowPlan but the loss function is changed to be the behavior cloning loss to maximize likelihood of human demonstrations.

In Figure 4 (a), we compare our method FlowPlan to the baseline method Polynomial Frenet, several ablations, and a model with the same architecture as FlowPlan but using an imitation learning (IL) loss against the expert driving demonstration. For the ablations, we examine taking samples directly from the latent space of the pretrained σVAE without the flow model. Samples are drawn from $\mathcal{N}(0,1)$ in the latent space. We also examine the NAF model without the σVAE to evaluate the efficacy of learning a normalizing flow mapping on the full control trajectory dimension without utilizing the expert demonstration conditioned latent space.

At low sample counts, FlowPlan significantly outperforms the baseline method Polynomial Frenet. This is because FlowPlan is better able to take advantage of historical and scene contextual information unlike the baseline which requires the use of hand coded rules to adapt to the context. As the number of samples increase the methods begin to converge to the same average min cost since the coverage of the action space is much broader at higher sample counts in the baseline, demonstrating that context matters less in the regime where coverage is high.

The σVAE model also performs better than the Polynomial Frenet baseline at low sample counts. We argue this is because the future AV control trajectory for most road network scenes is highly dependent on the historical dynamics information of the AV itself, which the model has access to. FlowPlan improves this performance by additionally accounting for the perceptual information. NAF without σVAE tends to produce non-smooth control trajectories as it cannot exploit a lower dimensional latent space to produce reconstructions of expert demonstrations which results in a higher average control loss. The model trained with IL loss performance does not depend on the sample count and performs worse than σVAE despite both models only having access to the expert driving demonstration. We argue this is due to the IL model learning a narrower distribution around a single expert trajectory given the context than σVAE which only has access to the AV dynamics history. Since the IL distribution is narrow around a single example there is less chance that a diversity of samples will produce meaningful differences in planner cost.

NAF and the NAF+σVAE+IL baselines rely on complete trajectory reconstruction. They have significantly worse performance than FlowPlan in both low and high sample regime. This shows the effectiveness of reasoning in a latent space of expert-like trajectories as used in FlowPlan. σVAE and context σVAE with behavior cloning perform better than the Polynomial frenet method under limited sample budget. FlowPlan outperforms both of these baselines while giving additional benefit of further refinement as shown in section V-B. In general it is difficult to compare imitation based method and cost function based methods on a common metric. To facilitate such a comparison we show a table M in Appendix of commonly used motion planning metrics that are essential for a good driving experience.

B. Target Distribution Learning

In this experiment, we are interested in empirically verifying if high probability density regions of the FlowPlan output distribution correspond to low planner cost. We propose finding high-probability regions in the output distribution of FlowPlan using the cross-entropy method (CEM). In CEM, we sample n times from an initial sampling distribution. The top e samples with highest probability density under output distribution from FlowPlan are selected and used to update the mean and the variance of the original sampling distribution. After N iterations of refinement we output the mean of the resulting sampling distribution as our latent variable which has the maximum density under the output distribution.

$$\begin{aligned} A_i = \{z^i\}, A_i \sim \mathcal{N}(\mu^m, \Sigma^m) \, \forall i \in n \\ A_{\text{elites}} = \; & \text{sort}(A_i)[-e :] \\ \mu^{m+1} = \; & \alpha * \text{mean}(A_{\text{elites}}) + (1-\alpha)\mu^m \\ \Sigma^{m+1} = \; & \alpha * \text{var}(A_{\text{elites}}) + (1-\alpha)\Sigma^m \end{aligned} \tag{9}$$

In Figure 4 (b), we show that with each iteration of CEM we sample higher probability trajectories in the FlowPlan output distribution and on evaluation of these trajectories we find the average planner cost decreases. This shows that high probability control trajectories under our learnt distribution correspond to low costs in the planner cost manifold. We can further use this method to improve our performance as shown in Appendix C2.

VI. Discussion

We present FlowPlan, a normalizing flow approach for generating control trajectory samples and associated probability density under the planner cost surface for AVs. As the flow model is connected to a learned perception & prediction model, which generates interpretable motion forecasting, the model leverages the full scene context during inference and adds little computational overhead to the existing AV stack. We compare this model using a dataset of real-world driving examples and show this approach is more efficient per sample than alternative approaches. We believe our method, FlowPlan, will perform similarly on other large real-world datasets [36], [37], as they are similar to our HES-4D dataset.

Limitations and Future Work. Because the model learns a non-transparent mapping from the prior distribution to the target, in order to ensure that safety maneuvers, such as max braking, are always in the considered trajectory set these have to be added in through an outside process. Additionally, the SDV trajectory samples are generated independently from the motion forecasting of other actors the predicted actions of other actors in the scene are not conditioned on the SDV intent. In the future we would like to extend this work to a unified probabilistic generative model that samples the SDV trajectory samples jointly with the motion forecasts of other actors.

References

[1] M. Bansal, A. Krizhevsky, and A. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," *arXiv preprint arXiv:1812.03079*, 2018.

[2] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011.

[3] C.-W. Huang, D. Krueger, A. Lacoste, and A. C. Courville, "Neural autoregressive flows," in *ICML*, 2018.

[4] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," in *Advances in neural information processing systems*, pp. 4743–4751, 2016.

[5] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," pp. 987 – 993, 06 2010.

[6] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2015.

[7] D. H. Jacobson and D. Q. Mayne, "Differential dynamic programming," 1970.

[8] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems.,"

[9] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[10] D. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *NIPS*, 1988.

[11] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4693–4700, 2018.

[12] X. Liang, T. Wang, L. Yang, and E. P. Xing, "CIRL: controllable imitative reinforcement learning for vision-based self-driving," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VII*, 2018.

[13] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, 2019.

[14] P. Tigas, A. Filos, R. McAllister, N. Rhinehart, S. Levine, and Y. Gal, "Robust imitative planning: Planning from demonstrations under uncertainty," 2019.

[15] A. Bhattacharyya, M. Hanselmann, M. Fritz, B. Schiele, and C.-N. Straehle, "Conditional flow variational autoencoders for structured sequence prediction," *ArXiv*, vol. abs/1908.09008, 2019.

[16] K. Mangalam, H. Girase, S. Agarwal, K. Lee, E. Adeli, J. Malik, and A. Gaidon, "It is not the journey but the destination: Endpoint conditioned trajectory prediction," *CoRR*, 2020.

[17] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[18] L. Sun, W. Zhan, and M. Tomizuka, "Probabilistic prediction of interactive driving behavior via hierarchical inverse reinforcement learning," pp. 2111–2117, 11 2018.

[19] C. You, J. Lu, D. Filev, and P. Tsiotras, "Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning," *Robotics and Autonomous Systems*, vol. 114, pp. 1 – 18, 2019.

[20] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, 2012.

[21] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning.," in *Aaai*, vol. 8, pp. 1433–1438, Chicago, IL, USA, 2008.

[22] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," *arXiv preprint arXiv:1710.11248*, 2017.

[23] M. J. Kochenderfer, *Decision making under uncertainty: theory and application*. 2015.

[24] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, "A tutorial on energy-based learning," 2006.

[25] B. Yang, W. Luo, and R. Urtasun, "Pixor: Real-time 3d object detection from point clouds," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

[26] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun, "Implicit latent variable model for scene-consistent motion forecasting," 2020.

[27] D. J. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.

[28] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," pp. 1094–1099, 06 2015.

[29] P. Martin, R. Murray, and P. Rouchon, "Flat systems, equivalence and trajectory generation," 01 2003.

[30] O. Rybkin, K. Daniilidis, and S. Levine, "Simple and effective vae training with calibrated decoders," 2020.

[31] G. Papamakarios, I. Murray, and T. Pavlakou, "Masked autoregressive flow for density estimation," in *NIPS*, 2017.

[32] B. Eysenbach and S. Levine, "If maxent rl is the answer, what is the question?," *arXiv preprint arXiv:1910.01913*, 2019.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[34] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, "Lasernet: An efficient probabilistic 3d object detector for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12677–12686, 2019.

[35] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun, "Jointly learnable behavior and trajectory planning for self-driving vehicles," *arXiv preprint arXiv:1910.04586*, 2019.

[36] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[37] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *CVPR*, 2020.

[38] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data," in *CoRL*, 2018.

[39] S. Casas, C. Gulino, R. Liao, and R. Urtasun, "Spatially-aware graph neural networks for relational behavior forecasting from sensor data," *arXiv preprint arXiv:1910.08233*, 2019.

Appendix

Supplementary Material for Imitative Planning using Conditional Normalizing Flow

A. Experiment Details

Expert Dataset: HES-4D contains more than one million frames collected over several cities in North America with a 64-beam, roof-mounted LiDAR. The labels are precise 3D bounding box tracks with a maximum distance from the self-driving vehicle of 100 meters. There are 6500 snippets in total, each 25 seconds long. We have access to high definition maps capturing the geometry and the topology of each road network in every city. Following previous works in joint perception and motion forecasting [38], [39] we consider a rectangular region centered around the self-driving vehicle that spans 144 meters along the direction of its heading and 80 meters across. This dataset involves trajectories observed in various situations like Lane Keeping, Merging, Intersections among others. Each trajectory is trimmed to 4 second blocks.

B. Model Details

σ-VAE: We use a conditional variational auto-encoder to compress the trajectory to a small latent space of 5 dimensions. This is motivated by the fact that the trajectories feasible under the kinodynamic constraints of the AV are limited and lie in a much smaller latent space. The context used for the conditioning is the 2 second history and is used to reconstruct the trajectory for the other 2 seconds of the trajectory obtained from the step above. Note that trajectories in these case are represented as control inputs of acceleration, steering pair and not the position-angle form. Rather than hand-tuning a desired weight between reconstruction error and KL divergence with prior in the VAE loss, we use σ-VAE which allows for this tuning to happen automatically. For the encoder and decoder, we use 3 convolutional layers with batch normalization followed by 2 fully connected layers.

Normalizing flow: We use the deep sigmoidal flow variant of the Neural Autoregressive flow [3] in this work. Our flow module comprises on 3 fully connected layers with 256 neuron units with exponential linear unit(elu) non linearities. The latent space of 5 dimensions obtained after passing the input through the encoder is transformed into a multimodal latent sampling distribution for low cost trajectories.

C. Imitation Learning Architecture

In this section we present the imitation leaning (IL) architecture used for experiments in section V-A. The exact details of the models is exactly same as described in section B. Our IL framework uses different model architectures during training and evaluation.

1) Training: The goal here is to learn from expert (human) demonstrations given the scene context. Since we are not costing any trajectories we do not need any detections, prediction and differentiable costing modules. We pass the expert trajectory through the cVAE Encoder, conditioned on AV history, to model the expert trajectory in lower dimension latent embedding referred to as Z_N. In reference to NAF, Encoder encodes the expert trajectory in a complex (multimodal) distribution. Forward NAF, conditioned on scene condition, maps the $Z_0 = NAF(Z_N)$ to a normal distribution. We train this model using maximum log-likelihood loss.

2) Evaluation: The goal here is sample trajectories of what most likely human would do given the scene condition, as compared to FlowPlan where goal is sample trajectories which minimize the cost functions. The architecture here is same as FlowPlan except the NAF model used here is inverse of NAF model used for training, $Z_N = NAF^{-1}(Z_0)$. Here the flow model provides a mapping from normal distribution to a complex distribution, conditioned on the scene. We pass the Z_N through cVAE Decoder to obtain trajectories from the latent variable. We cost the trajectories with a differentiable costing module to find the best trajectory under planner cost surface for the given scene.

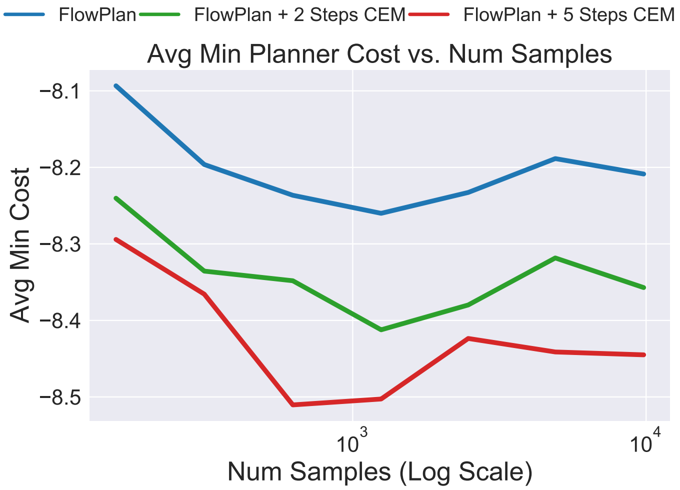


Fig. 5. Planner cost decreases with increase in CEM iterations on FlowPlan.

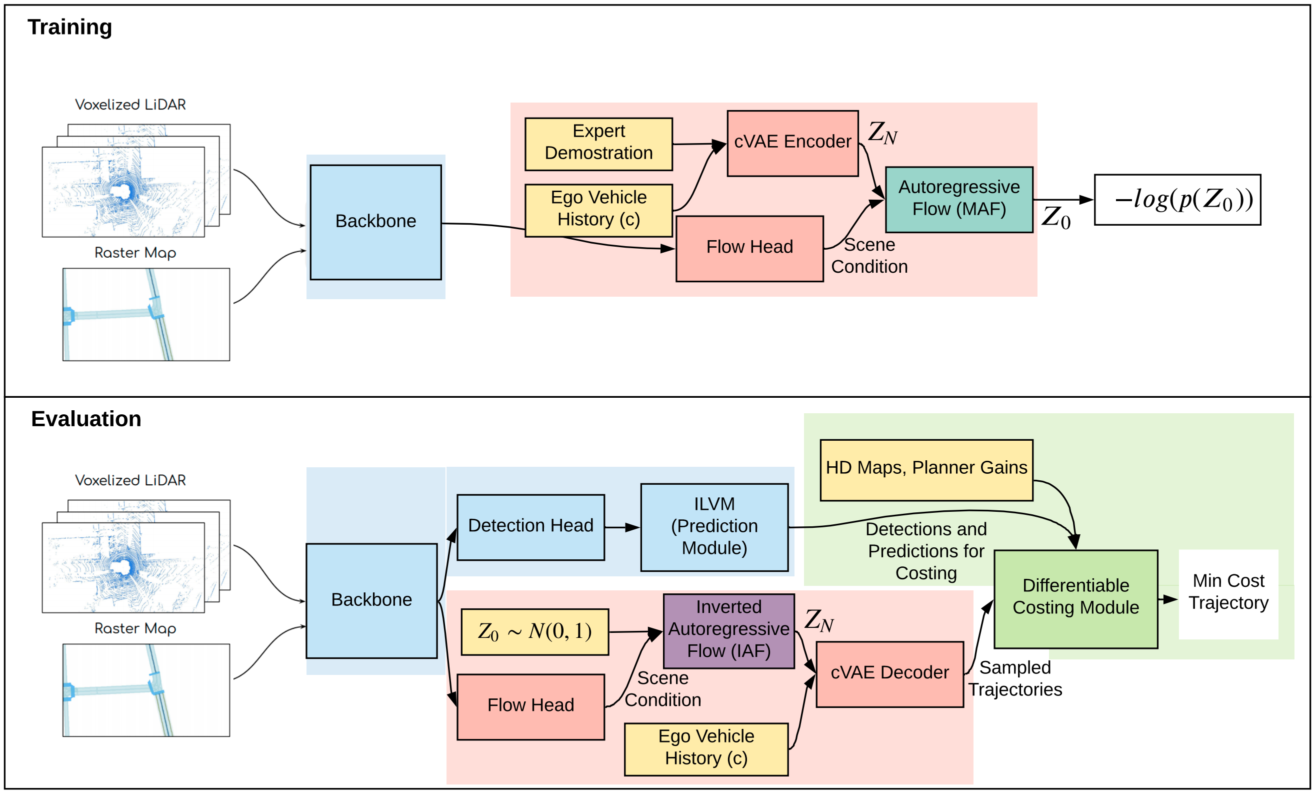


Fig. 6. Imitation Learning Architecture.

FlowPlan provides us with a good initial sampler producing trajectories which have low costs. Generally planners on AV work under time constraints, and have a fixed amount of available time budget for planning. For example, a typical planner aims to produce plans at 10 Hz frequency.

In this section we study improving upon the already obtained plans when we have additional time budget available.

To achieve this we rely on a similar method as used in Section V-B. We use CEM to refine our solutions iteratively. We sample initially using FlowPlan and successively improve upon the solution using the Planner Cost (in contrast to the Flow density) as the CEM objective now. We experiment with 2 and 5 iterations of CEM and find that increasing CEM iterations increases the quality of solution. For a fair comparison against the basic FlowPlan we sum up the number of samples used during all the CEM iterations (x-axis of Figure 5).

In this section we share some results which give more insights into our σ-VAE's latent space.

D. Gaussian Sampling

We learn a latent embedding using a diverse set of trajectories obtained from expert demonstrations. Figure 7 shows some examples fo the trajectories learned by the VAE. It also shows that the trajectories produced by the VAE may not be low cost and can be compared to Figure 3 where we see that flow learns a tight low cost distribution in this latent space.

E. Latent Space Interpolation

Figure 8 shows the trajectories we obtain when interpolating in the latent space.

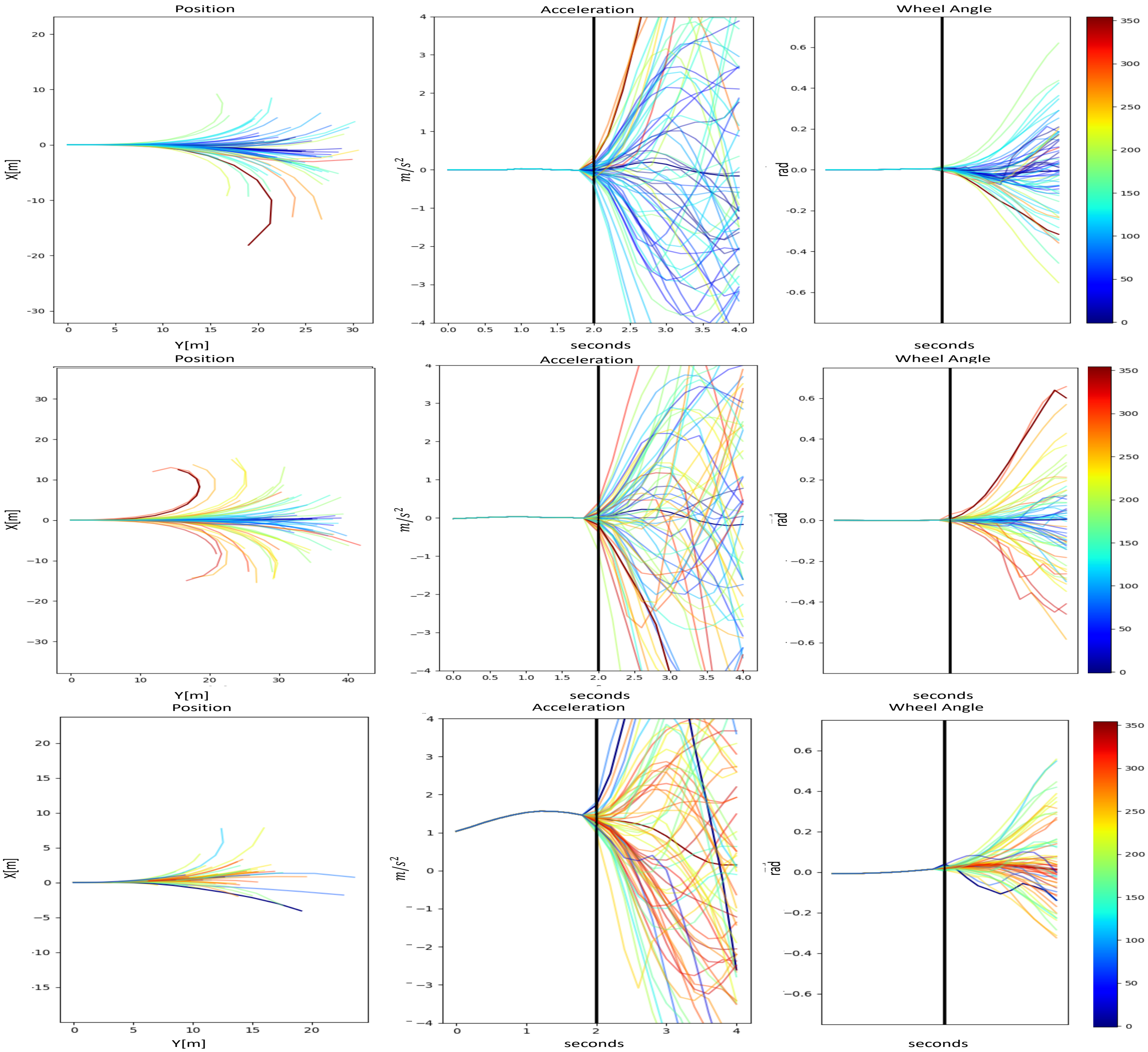


Fig. 7. Output of σ-VAE when sampling from $N(0,1)$ from the latent space. This demonstrates that naively sampling from $N(0,1)$ results in control trajectories with high coverage of the action space but not necessarily in low cost trajectories under the planner cost surface. Colors here illustrate the respective cost of the control trajectory.

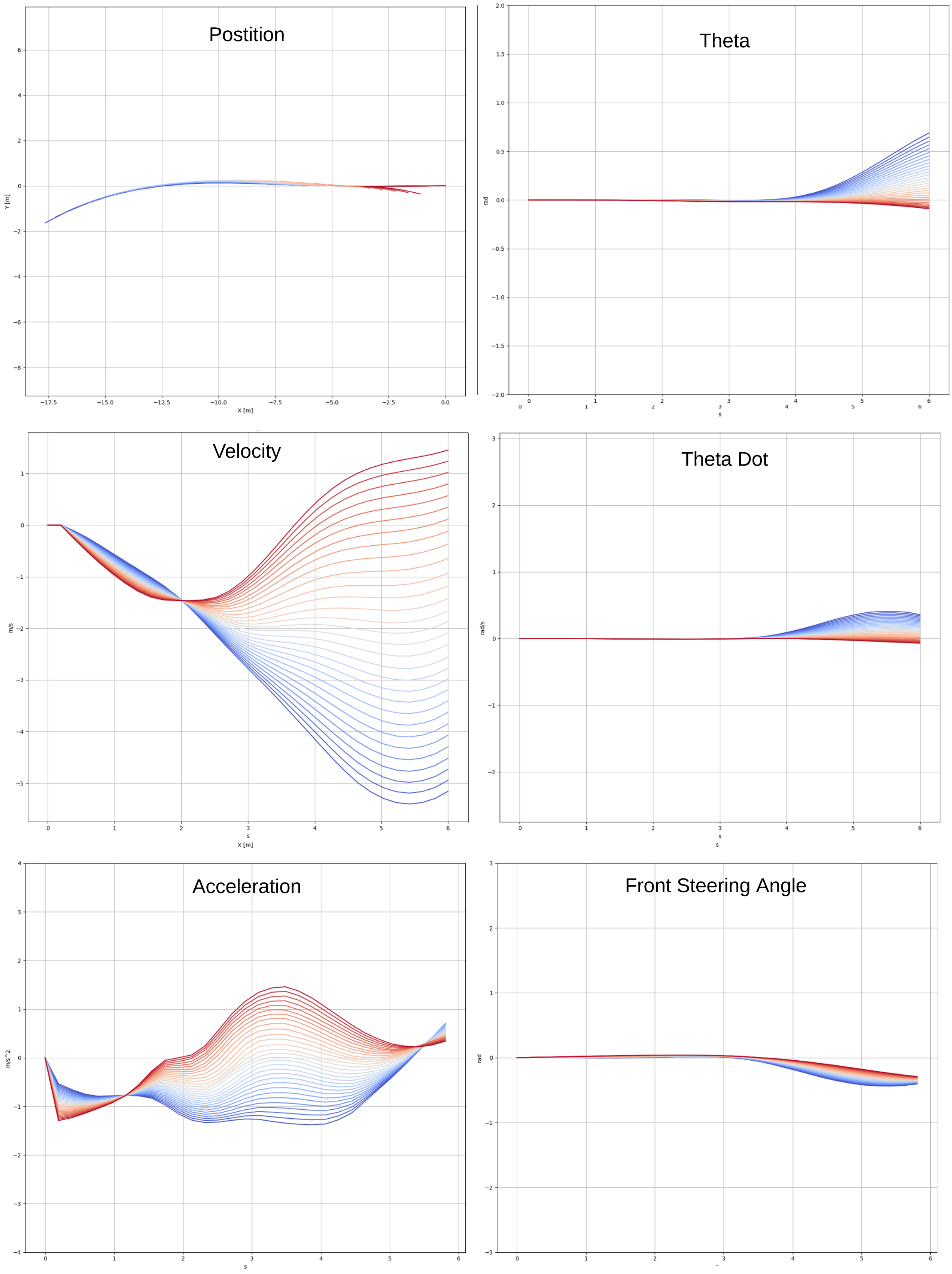


Fig. 8. Smooth trajectories from σ-VAE's latent space. Here we linearly interpolate between latent variables, from red to blue, belonging to the actual dataset and show that the new latent variables result in smooth output. Theta is the heading of the AV.

F. σ-VAE Latent Space Analysis

In this section we experiment with different latent dimensions for FlowPlan. A smaller dimension makes it harder for the latent space to capture the high dimensional trajectory distribution and a high dimensional latent space increases the complexity of learning and generalization. We find that a latent dimension of 8 achieves the minimum average cost when tested on the HES-4D dataset. In Figure 9, we see that the average cost of FlowPlan decreases as we increase the latent dimension from 3 to 8 and then increases as the dimensions are increased to 32.

KL divergence is a good metric to evaluate if the sampling policy we learn is close to the Boltzmann distribution induced by the cost surface. We compute the KL divergence using a sample based estimate, but we need a partition function since our cost functions are unnormalized. We rely on importance sampling to compute a sample based estimate of the partition function, where the importance sampling distribution is our current flow distribution. The importance sampling distribution becomes optimal as the policy approaches the Boltzmann distribution induced by the cost functions.

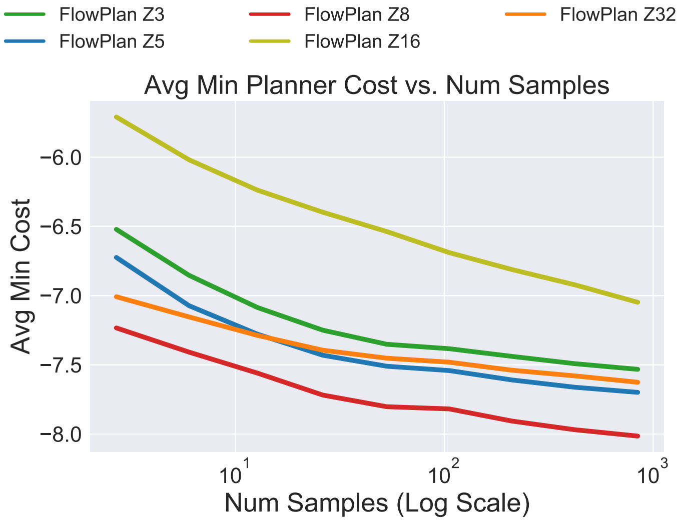


Fig. 9. Latent Dim Comparison

$$Z = \int e^{-C(\tau)} d\tau$$
$$\hat{Z} = \mathbb{E}_{z \sim Z_N}\left[\frac{e^{-C(decoder(Z_N))}}{p(Z_N)}\right] \tag{10}$$

where C is the unnormalized cost function, which costs the entire trajectory following Appendix F.

A sample based estimate for the reverse-KL divergence can be obtained as follows:

$$\begin{aligned} D_{\mathrm{KL}}[q(z_N|\theta, b) \mid\mid J(z_N|b)] &= \mathbb{E}_{z_N \sim Z_N}\left[\log q(z_N|\theta, b) - \log \frac{e^{-C(decoder(Z_N))}}{Z}\right] \\ &= \mathbb{E}_{z_N \sim Z_N}\left[\log q(z_N|\theta, b) + C(decoder(Z_N)) + \log \hat{Z}\right] \end{aligned} \tag{11}$$

The decoder takes the latent embedding of a trajectory and outputs the reconstructed trajectory.

In this work we generate control trajectories in the space of steering angle and acceleration. These control trajectories will be simulated by a forward bicycle model to obtain trajectories in *frenet* space [5] and Cartesian space. Frenet space represents the trajectory of a car as latitudes and longitudes based on a nominal path that a car is expected to follow at each points. We will consider trajectories in both frames and the predictions obtained for each actor in the scene as a interpretable belief for costing purposes.

We use a set of costs that allow for safe driving with user comfort in mind. In particular our cost functions can be divided into five costs: path distance, centerline, obstacle collision, jerk, and twist costs. We expand upon all the cost functions in detail below:

G. Path Distance Cost C_d

A basic objective of the car is to move along the directed centerline. A *centerline* is defined as a nominal path safe to follow in presence of no obstacles. It is obtained as a part of the High Definition maps and in this work is just the centerline equidistant from the two lane boundaries. We reward the agent to cover as much distance on the centerline as possible. The cost function for the distance cost looks as follows:

$$C_d = -\text{Agent cartesian displacement projected on the centerline} \tag{12}$$

H. Centerline Cost C_c

In addition to travelling as much longitudinal distance as possible we would like our AV to stay close to the centerline. We do this be penalizing the normal displacement of a cartesian trajectory to the centerline along each control point along the trajectory.

$$C_c = \sum_{t=1}^{T} d_t^2 \tag{13}$$

where d_t is the normal distance (latitude) of the t timestep in trajectory to the centerline.

I. Obstacle Collision Cost C_o

Penalizing collisions is an important aspect to ensure safety of AV. We consider the future actor predictions in the scene and unrolled trajectory of the AV using a dynamics model to check for possible collisions. The cost function is summed over all the probabilistic elements of the scene using the probabilities output by the prediction module. The expected cost is passed through a rectified linear unit to penalize cost for only those actors that enter a 3 meter radius of the car.

$$C_o = \sum_{obstacles} \text{prob}_{obs} * (rectified(\text{distance to obstacle}))^2 \tag{14}$$

J. Jerk Cost C_j

Jerk is defined as rate of change of acceleration. Minimizing jerk is crucial to obtain a comfortable user experience. We encode this directly in our cost functions.

$$C_j = \sum_{t=1}^{T-1} (\dot{a})^2 \tag{15}$$

where a is acceleration and $\dot{a}$ is the jerk (first derivative of acceleration).

K. Twist Cost C_t

Ensuring smooth changes in curvature for the trajectory is another aspect of encoding user preference for comfort. We do this by directly imposing a smoothness constraint on the steering of the vehicle.

$$C_t = \sum_{t=1}^{T-1} \dot{c}_i^2 \tag{16}$$

where c is curvature and $\dot{c}$ is the twist (first derivative of curvature).

L. Cost Gains

The final cost for a trajectory is the weighted combination of the cost functions applied to the trajectory by cost gains. The gains are manually tuned for best performance and interpretability.

$$\textbf{Final Cost} = w_d * C_d + w_c * C_c + w_o * C_0 + w_j * C_j + w_t * C_t \tag{17}$$

M. Motion Planning Metrics

Models	Avg. Jerk (mpsss)	Avg Lat Accel (radpss)	Avg. Progress (m)	Collision 0.5s	Collision 1.0s	Collision 1.5s	Collision 2.0s
Human	4.14	1.93	13.05	0.00	0.00	0.00	0.00
Polynomial Frenet	3.03	3.89	13.27	0.00	0.00	0.01	0.02
NAF	1.59	4.10	10.44	0.00	0.00	0.03	0.11
σ-VAE	3.47	3.20	13.29	0.00	0.00	0.00	0.1
FlowPlan	1.53	3.05	13.21	0.00	0.00	0.00	0.1

Fig. 10. Motion Planning Metrics: We compare the performance comparison of different sampling techniques used for generating low cost control trajectories on various planning metrics. This table provides an intuition into how FlowPlan was able to generate plans with lower cost. Specifically, plans generated by FlowPlan has lower Avg Jerk, Avg. Lat Accel, higher Avg. Progress and minimal collisions.