Self-Supervised Simultaneous Multi-Step Prediction of Road Dynamics and Cost Map

Elmira Amirloo[1], Mohsen Rohani[*1] , Ershad Banijamali[1], Jun Luo[1], Pascal Poupart[2]
[1]Noah's Ark Lab, Huawei, Toronto, Canada
[2]School of Computer Science, University of Waterloo, Waterloo, Canada
{elmira.amirloo,mohsen.rohani,ershad.banijamali1,jun.luo1}@huawei.com, ppoupart@uwaterloo.ca

Abstract

While supervised learning is widely used for perception modules in conventional autonomous driving solutions, scalability is hindered by the huge amount of data labeling needed. In contrast, while end-to-end architectures do not require labeled data and are potentially more scalable, interpretability is sacrificed. We introduce a novel architecture that is trained in a fully self-supervised fashion for simultaneous multi-step prediction of space-time cost map and road dynamics. Our solution replaces the manually designed cost function for motion planning with a learned high dimensional cost map that is naturally interpretable and allows diverse contextual information to be integrated without manual data labeling. Experiments on real world driving data show that our solution leads to lower number of collisions and road violations in long planning horizons in comparison to baselines, demonstrating the feasibility of fully self-supervised prediction without sacrificing scalability.

1. Introduction

Conventional autonomous driving (AD) stacks consist of various modules [30]. A perception component is responsible for detecting objects in the scene and a prediction module for projecting their positions in the future. Based on their outputs a motion planner generates a desired trajectory according to a manually specified cost function [6], which is in turn executed by a controller. A key advantage of this approach is the interpretability of the final decision. For example, in case of accident, each component can be investigated individually. However, with different parts designed and tuned separately, each module is not aware of the errors made by the other parts. In many cases, there is no clear way for estimating the model uncertainty and propagating it to the system. In addition to massive amount of human-labelled data required to train the perception components, the manual design and tuning of the cost function for motion planning tends to limit the system's ability for dealing with complex driving scenarios .

*The authors contributed equally.

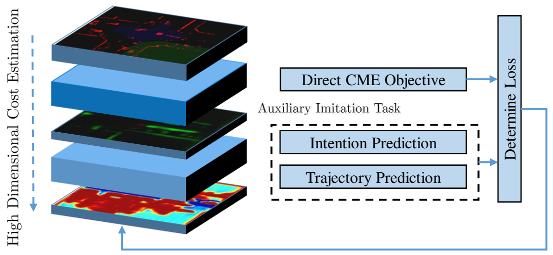

Figure 1. High Dimensional Cost Map Estimation.

As an alternative, several works [2, 7, 34, 5] proposed driving systems that use raw sensory input to directly produce control commands (i.e., acceleration and steering). This approach allows full backpropagation and eliminates the need for a cost function. Since a large quantity of data can be collected from cars equipped with appropriate sensors and directly used for training without human labelling, this approach is potentially highly scalable with data and compute. However, such a monolithic approach lacks internally interpretable components, offers little insight as to how system faults may arise, and is thus ill-suited for safety-critical real-world deployment.

In this paper, we propose a new approach that allows meaningful interpretation and avoids manual data labeling and design of cost function. Our approach is centered around a novel architecture for learning-based space-time Cost Map Estimation (CME). The proposed method is highly scalable as it can be trained in a fully self-supervised fashion. Moreover, the space-time cost map has both a natural interface with motion planner and an interpretable domain-specific semantics.

Specifically, our architecture encodes high dimensional Occupancy Grid Maps (OGM)s as well as other contextual information (e.g., drivable area and intersections) and simultaneously predicts the OGMs and estimate the cost map for multiple steps into the future. This leads to interpretable intermediate representations in the form of OGMs. A cost map (CM) is a grid map where the value of each cell represents the cost of driving into that cell location. By extending the predicted CM multiple steps into the future, we arrive at a sequence of space-time CMs. These CMs can then be used by a motion planner to rank possible future trajectories through integrating the cost over the cells these trajectories occupy.

Importantly, while it is obvious that OGM prediction training can be made self-supervised using sequences of driving data (e.g., [16]) so long as occupancy estimation is accurate, labels for self-supervised training of CM estimation are much harder to synthesize. To solve this problem, we decompose the CME objective into two parts. The first one injects the prior knowledge about the environment where it is available (e.g., occupied cells are high cost). However, there is no explicit information about the cost of most of the cells. Hence, we propose using an auxiliary task to guide the training. Using auxiliary objectives for improving the performance of a model in a primary task has been proven to be effective in different fields [13, 29, 3]. Similarly, in this work we define an auxiliary imitation task that forces the model to predict the expert's intention and trajectory based on the estimated CMs. For this task a data-driven set of intentions capturing different modes of driving is used. This objective term pushes the model to fill in the blanks and arrive at complete and systematically accurate predictions of the CM.

The main contributions of this paper are as follows:

- An architecture for estimating CMs simultaneously with OGM prediction from human driving data that is fully self-supervised and requires no extra data labeling,
- A set of specific training objectives that combines environment constraints, expert's behavior, map information as well as solving auxiliary imitation tasks leading to estimating space-time cost maps,
- An empirical demonstration of the effectiveness of this design in the overall performance of an AD system through multiple experiments and generalization tests.

2. Related Work

Unlike the monolithic end-to-end approach, our proposal replaces modules of a conventional AD stack to enable doing a fully self-supervised learning of cost maps. Our main goal is balancing scalability and interpretability. From this perspective, we discuss how our work compares and contrasts with existing works.

2.1. Cost Design and State Prediction

Conventional motion planners typically optimize trajectories according to a predefined cost function [32, 25, 11]. However, manually defining and tuning a cost is extremely hard for highly dynamic environments such as driving. This has led to trade-offs where AD solutions largely avoid difficult interactions. More recently learning based received significant attention in planning [17]. Similarly, we aim at replacing the manual cost function with learning-based space-time CMs that predict into the future. However, it is important to note that our proposal is compatible with both classical and learning-based planning methods insofar as they can use the predicted CMs to rank candidate trajectories.

Prediction is a crucial part of any AD stack. Learning-based prediction is increasingly popular in AD research. In [14], detection, tracking and motion forecasting are done using a single network. [24] uses a bird's eye view image of navigation and the motion history of agent to predict its future path. [28, 21, 20] combine high dimensional and low dimensional data to predict multi-modal trajectories. Both video prediction [34] and OGM prediction [10, 16] have been done in AD research. In contrast to such work, our approach aims to simultaneously predict states and estimate CMs. In many cases, there is no clear way to compute the uncertainty of the isolated prediction models and propagate it to the motion planner. Furthermore, although we evaluate the quality of estimated CMs with a specific trajectory sampler, in our design any set of trajectories can be ranked using the estimated CMs. Predicting trajectories directly makes both generalization and adaption to a new or temporary constraint harder.

2.2. RL and IRL

As a powerful framework for sequential decision making, reinforcement learning (RL) makes an attractive choice for AD research. [19] proposed a hierarchical RL scheme that divides tasks into high-level decisions and low-level control. [26] introduced a two-phase approach for cruising and merging tasks in autonomous driving, where future states are first predicted from the current ones and then an RL planner uses these predictions to output acceleration. These proposals utilize low-dimensional states derived from perceptual processing of high dimensional sensory data. However, by implicitly conflating the processed low dimensional state estimations as the actual states, error and uncertainty do not get propagated through to improve the whole system. Moreover, similar to the case of manual design of the cost function, defining reward for RL solutions remains an open research challenge.

Inverse reinforcement learning (IRL) focuses on learning a reward function from the expert's behavior, with maximum entropy [36] being a popular method. However, standard IRL algorithms can not deal with high dimensional data and continuous space. Some works such as [23] extended these algorithms to use high dimensional data. But linear estimation of reward function can adversely limit generalization of the system. Similar to our work, [33] estimates a high dimensional cost map using a maximum entropy deep IRL framework. There are two main differences between this work and ours. First, the system is not modular which challenges the interpretability in case of failure. Second, in such IRL frameworks training the policy and estimating the reward is done together. Empirically training the new policy with the learned reward does not lead to similar performance. Hence, if the set of actions or the policy model needs to be changed, the learned reward may not lead to similar performance.

2.3. Imitation Learning

Imitation learning (IL) is a standard approach to learning from expert demonstration. It has the advantage of not requiring costly online exploration typically necessary for RL methods. [2, 7] follow a monolithic, end-to-end approach where the network receives images and generates control commands. While this approach avoids costly data labeling, direct behavior cloning suffers from cascading errors when dealing with out-of-distribution inputs. To alleviate this issue, [12] adopted Generative Adversarial Imitation Learning (GAIL) [9]. But the interpretability challenges remain.

Since highly mature control solutions are widely used in AD systems, learning to do level control is often unnecessary and learning based trajectory planners are more practical. [1] uses a modular network to generate driving trajectories. They propose novel data augmentation approaches to generate scenarios such as collisions which most real-world datasets lack and give the network a better opportunity to learn to handle such situations. [35] also has a modular design where the perception module does 3D object detection and motion forecasting. A cost volume generation is done by another component of the overall network. They use the cost volume to choose a trajectory with the lowest cost. However, in contrast to our proposal, both of these works heavily rely on labeled data for the perception components.

3. Technical Approach

We address the problem of predicting a high dimensional cost map by proposing a modular architecture that can be trained end-to-end in a self-supervised fashion. We then use this prediction to evaluate and score different trajectories. The model takes a sequence of LiDAR point clouds and other contextual information (e.g. map) as input. It predicts the future OGMs representing road dynamics and estimate the space-time cost of driving in each cell of the OGM simultaneously. The proposed architecture has two components: (1) an OGM predictor and (2) a CM estimator. Note that this design adds interpretability because the predicted OGMs over the planning horizon are independently semantically meaningful.

3.1. Input Encoding and Network Architecture

On the input side, frames of LiDAR point clouds from the immediate past are first converted into a sequence of OGMs. These OGMs are then transformed to a reference frame attached to the vehicle's current position. Both prediction and cost estimation are done with respect to this coordinate system to avoid unnecessary complexity due to motion of the ego vehicle. Moreover, similar to [1], we encoded semantic information from the HD map such as drivable area and intersection structure in separate channels. We then concatenated the map encodings with OGMs to form the input to the network.

Predicted OGMs are represented by a binary random variable $o_k(i,j) \in \{0,1\}$ where $o_k(i,j)$ is the occupancy state of the cell at the i^{th} row and the j^{th} column at the k^{th} time step, with 1 for occupied and 0 for empty. p_o is the probability of occupancy associated with each cell. Cost value at each cell $c_k(i,j)$ is coded similarly where 1 and 0 are assigned to high and low cost cells respectively.

We study two alternative architectures. For the *Recurrent Cost Map Estimator* (RCME), we assume the CMs in each time step are conditionally dependent on the information in previous time steps. For the *Multi-Step Cost Map Estimator* (MSCME), we omit this assumption. In Section 4 we show that these two architectures have similar performance for shorter prediction horizons. The recurrent architecture performs better for longer planning horizons.

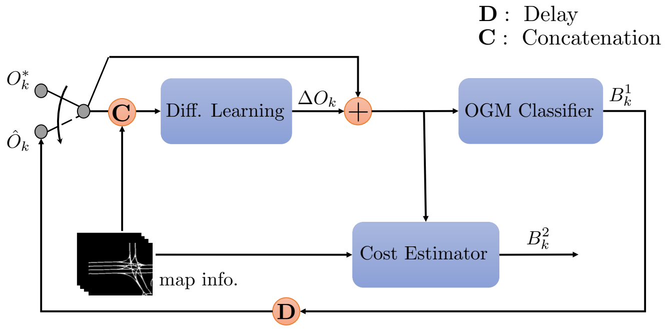

Figure 2. Recurrent Cost Map Estimator

3.1.1 Recurrent Cost Map Estimator

The RCME incorporates the *difference learning* method in [16] for OGM prediction and extends it with our cost estimator module to simultaneously predict CMs (Figure 2). Formally, the output of the network at time step k can be represented as a two-channel tensor:

$$\mathbf{B}_k^1 = \left[p_o\big(o_k(i,j)\big)\right] = \begin{bmatrix} p_o\big(o_k(1,1)\big) & \dots & p_o\big(o_k(1,W)\big) \\ \vdots & \ddots & \vdots \\ p_o\big(o_k(H,1)\big) & \dots & p_o\big(o_k(H,W)\big) \end{bmatrix} \quad (1)$$

$$\mathbf{B}_k^2 = \left[p_c\big(c_k(i,j)\big)\right] = \begin{bmatrix} p_c\big(c_k(1,1)\big) & \dots & p_c\big(c_k(1,W)\big) \\ \vdots & \ddots & \vdots \\ p_c\big(c_k(H,1)\big) & \dots & p_c\big(c_k(H,W)\big) \end{bmatrix} \quad (2)$$

where B_k^1 and B_k^2 are the first and second output channels respectively. $o_k(i,j)$ is taken to be independent of the values of other cells at time step k, but conditioned on values of all cells in previous time steps. And the same assumption is made for $c_k(i,j)$:

$$p_o\big(o_k(i,j)|\mathcal{O}_{k-1}, \mathcal{O}_{k-2}, ...\big) \quad (3)$$

$$p_c\big(c_k(i,j)|\mathcal{C}_{k-1}, \mathcal{C}_{k-2}, ...\big) \quad (4)$$

where

$$\mathcal{O}_k = \big\{o_k(m,n)|m = 1, ..., H; n = 1, ..., W\big\} \quad (5)$$

$$\mathcal{C}_k = \big\{c_k(m,n)|m = 1, ..., H; n = 1, ..., W\big\} \quad (6)$$

and m and n are indices ranging over the entire OGM. The conditional probabilities in Equation 3 and 4 may be captured using the recurrent architecture proposed. In practice, a short history suffices. The network observes OGMs for the past τ time steps and predicts the OGMs for the next T time steps while estimating a CM at every step.

In reality not every cell in the OGM changes between two time steps. The **Difference Learning** module implicitly distinguishes between dynamic and static objects. By adding the features extracted by this module to the previous observed OGMs B_k^{1*} or predicted OGMs $\hat{B}_k^1$ and stacking them with the encoded map, the **OGM Classifier** can be trained to effectively and efficiently predict if a cell is occupied or not. We did not use the same architecture for estimating CMs as they are not directly *observable* and imposing such a feedback loop can amplify error in CM estimation. Hence, the stacked OGM features are separately fed along with the encoded map to the **Cost Estimator** module that consists of an encoder and a decoder. The encoder has {32, 64} 3×3 convolution filters with stride 2. The decoder has two deconvolution layers with {64, 32} 3×3 filters with stride 2 each deconvolution layer is followed by a convolution layer with the same size and stride 1.

3.1.2 Multi-Step Cost Map Estimator

The MSCME architecture is illustrated in Figure 3. Similar to RCME, the OGMs and the encoded maps are fed to the predictor network. The predicted and observed OGMs are then stacked with the encoded map and passed through an encoder and then a decoder to estimate a CM for T time steps. To avoid computationally expensive 3D convolutions we concatenate time steps along the last dimension. In order to get similar performance as the previous architecture we used {32, 64, 128} filters in the encoder and {128, 64, 32, T} filters in the decoder where T is the number of time steps we predict the CM for.

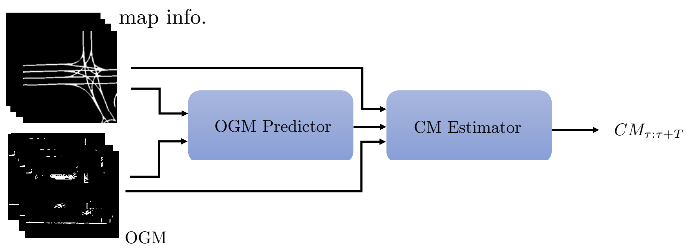

Figure 3. Multi-Step Cost Map Estimator

3.2. Training Loss

The designed architecture is a multi-task network. The objective function is accordingly defined to direct the network to learn each task:

$$\mathcal{L}_{total} = w_1\mathcal{L}_{Pred} + w_2\mathcal{L}_{CME} \quad (7)$$

where w_1 and w_2 are hyperparameters. We define each term in detail below.

3.2.1 Prediction Loss

We follow [16] to formulate OGM prediction as a classification problem where each cell can be occupied or not. Hence, the objective function includes a pixel-wise Cross-Entropy between the predicted OGMs, $\hat{B}^1$, and the target OGMs, B^{1*}, multiplied by a visibility matrix, $\mathcal{V}$, described in [8] to handle occlusion. Due to unbalanced number of occupied and free cells, we normalize the loss by the ratio of occupied/free cells, η. Finally, to push the predicted OGMs toward the target OGMs we use Structural Similarity Index Metrics (SSIM) [31]. The OGM prediction loss is then defined as:

$$\mathcal{L}_{Pred} = \frac{\eta}{WH}\sum_x\sum_y \mathcal{V}\odot\mathcal{H}(\hat{B}^1, B^{1*}) + \gamma(1 - SSIM(\hat{B}^1, B^{1^*})) \quad (8)$$

where H and W are the OGM dimensions, $\odot$ denotes the element-wise product, $\mathcal{H}(a,b)$ is the pixel-wise cross-entropy and γ is a hyperparameter.

3.2.2 CM Estimation Loss:

Since there is no ground truth for the CM, defining an objective function which pushes the network to learn meaningful CM is challenging. Relying only on the expert's trajectory makes it difficult for the network to generalize. The expert's trajectory only occupies a few cells and it does not give information about the most of the surrounding area. To

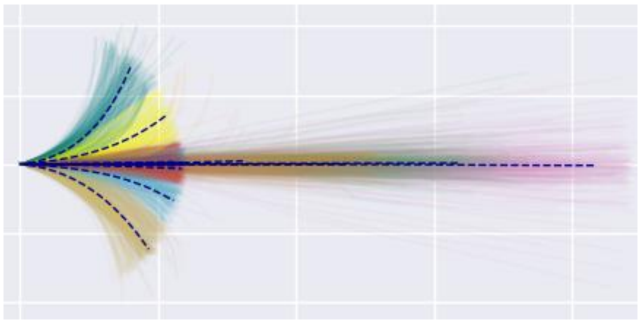
Figure 4. Different modes of driving driven from data. Dashed lines show the mean of each cluster.

address these issues we define an objective function consisting of two terms:

$$\mathcal{L}_{CME} = \alpha \mathcal{L}_p + \beta \mathcal{L}_{aux} \tag{9}$$

where α and β are hyperparameters.

$\mathcal{L}_p$ is defined to inject the prior knowledge about cell cost such as the high cost associated with non-drivable areas. Specifically, $\mathcal{L}_p$ is a classification loss comparing the generated CM and a target C_{target}, which at each time step is 0 for the cells occupied by the expert and 1 for non-drivable areas and the occupied cells in drivable areas. Since there is no information about the other cells we do not want to push the network to assign any values to them. Moreover, the number of cells belonging to the expert's trajectory (low cost cells) are far less in number than the high cost ones. In order to address both of these issues, we calculate the loss on a subset of cells selected by a mask, $\mathcal{M}$ with 0 and 1 elements. The total number of 1s is set to a predefined number, N. $\mathcal{M}$ elements are 1 for the pixels occupied by the expert. The rest of 1s are sampled from the high cost cells, i.e. cells occupied by objects or in non-drivable area, with the high cost cells that are occupied having 2 times more chance to be selected. This ratio empirically speeds up training. The L_p loss function is then:

$$\mathcal{L}_p = \frac{1}{WH} \sum_x \sum_y \mathcal{M} \odot \mathcal{H}(B_k^2, C_{target}) \tag{10}$$

We added an *Imitation Network* to the architecture and defined an auxiliary task in overall objective function, $\mathcal{L}_{aux}$, in order to indirectly push the CMs to be a representation of the underlying reason for the expert's behavior. For this purpose, a sequence of estimated CMs from time τ to the prediction horizon $\tau + T$ are fed to an encoder. These features are then utilized by an *intention prediction* head and a *a regressor head* to predict the expert's trajectory.

A predefined set of "intentions" is used to represent different semantic modes made by the expert (e.g. changing lane, speed up, slow down), $\mathcal{I} = \{i^k\}_{k=1}^K$ where $i^k = \{s_1^k, ...s_T^k\}$ defines a trajectory for T timesteps. These intentions are derived by clustering the expert's trajectories in the dataset (Figure 4). Specifically, we used the DBSCAN clustering algorithm and Hausdorff distance to cluster trajectories. Given the CMs, for each intention i^k, the intention predictor head predicts a Bernoulli distribution $p(i^k|CM_{\tau:T+\tau})$ to determine whether the expert chose that driving mode or not. Hence, each trajectory can belong to multiple clusters at the same time. In this way we do not penalize the network for choosing the modes that are close to each other. One can also use the soft labels in a cross-entropy setting where the labels are the normalized distances to clusters. However, empirically our problem formulation worked better for this architecture. The regressor head then outputs K offsets, s_k^o, between the mean of each cluster μ_k and the expert's trajectory s^*. We then used a weighted MSE to optimize the networks.

$$\mathcal{L}_{aux} = \frac{1}{K} \sum_k \mathcal{L}_{cls}(p_k, p_k^*) + \lambda \sum_k \omega_k MSE(\mu_k + s_k^o, s^*) \tag{11}$$

where p_k is the probability of an intention to be the expert's intention, ω_k is the normalized distance of the groundtruth trajectory to each mode and λ is a hyperparameter.

3.3. Motion Planning

To evaluate the quality of the CMs, we use them for motion planning. We follow [35] to use clothoids [27] as well as circular and straight lines to define the shape of candidate trajectories. The velocity profile of a candidate trajectory is determined by sampling acceleration in the range of $[-5, 5]$ m/s^2 and velocity between 0 and the speed limit. Since computing the cost of each candidate trajectory using the estimated CMs is a cheap operation, our motion planning module is computationally very efficient.

Note that the output of the regressor head for the imitation auxiliary task could in principle be used for trajectory planning. However, we opt for a simple sampling method that uses the CMs. This is partly to demonstrate the versatility of the CMs when working with motion planning methods and partly because the CM estimations are presumably much more reliable since they integrate by design broader concerns beyond imitating the expert.

4. Experiments

We applied our approach to the Argoverse [4] dataset. The LiDAR point clouds are converted to 256 × 256 BEV with the ground removal described in [15]. Moreover, to increase the ability of our model in handling occlusions, we applied a visibility mask as in [8]. We also encoded the information from the map into 8 different channels.

We report performance on 3 different settings for the

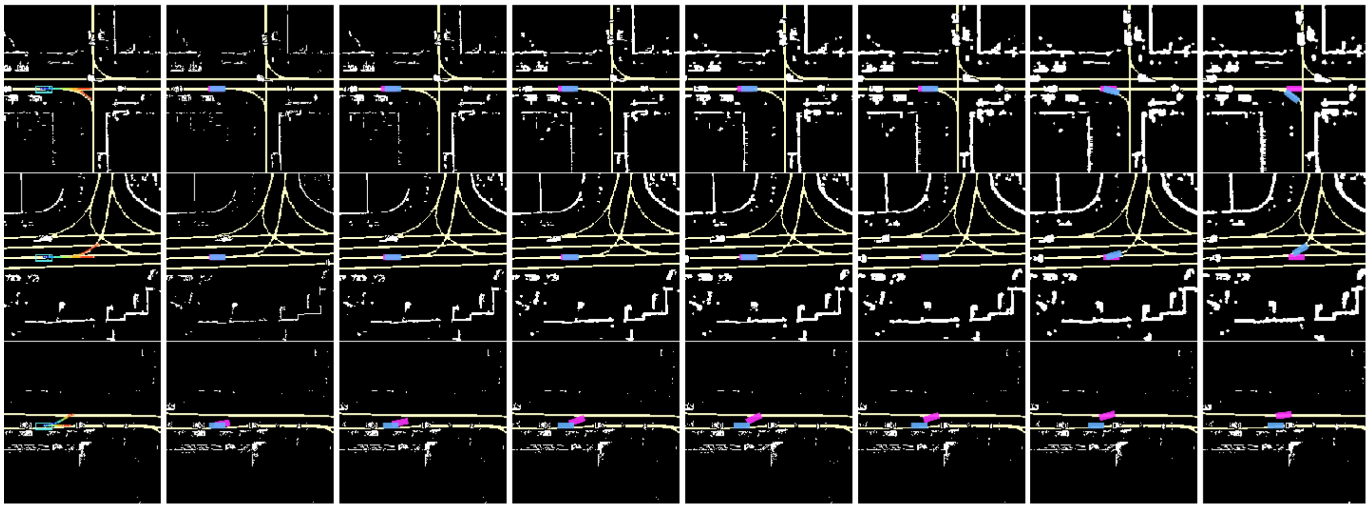
Figure 5. Low cost trajectories selected by the proposed method. In these experiments the algorithm is forced to pick the trajectories from different driving modes as described in Section 4.3.1. Each row, is a different scenario where the first column shows the planned trajectory. In next columns the car is moved according to that trajectory.

Algorithms		$\tau = 1$ sec $T = 1$ sec			$\tau = 2$ sec $T = 2$ sec			$\tau = 1$ sec $T = 3$ sec		
Alg.	Arch.	minADE	CR(%)	RV(%)	minADE	CR(%)	RV(%)	minADE	CR(%)	RV(%)
BC	BC-MLP	**0.05**	**0.00**	0.07	**0.79**	2.36	2.61	3.18	7.11	5.73
	BC-LSTM	0.08	**0.00**	0.09	0.84	2.57	1.87	3.01	6.19	5.03
RuleCM		1.03	0.34	**0.00**	2.21	3.18	**0.00**	3.24	4.93	0.09
MFP$_{K=6}$	MFP.1	0.21	**0.00**	**0.00**	1.92	1.18	0.86	3.78	4.33	2.98
	MFP.3	0.21	0.09	0.05	1.92	2.07	0.97	3.78	5.96	3.59
ESP$_{K=6}$	ESP.1	0.41	0.21	**0.00**	2.07	2.84	1.15	3.97	3.87	4.62
	ESP.3	0.41	0.94	0.36	2.07	2.92	1.42	3.97	5.46	4.93
CME (ours)	MSCME.a.1	0.15	0.01	**0.00**	1.94	0.92	0.01	3.52	1.68	0.05
	MSCME.b.1	0.11	**0.00**	**0.00**	1.67	0.85	**0.00**	3.28	1.57	**0.01**
	MSCME.b.3	0.11	0.01	**0.00**	1.67	0.91	**0.00**	3.28	1.62	**0.01**
	RCME.a.1	0.18	**0.00**	**0.00**	2.74	0.67	0.01	**2.92**	0.84	**0.01**
	RCME.b.1	0.17	**0.00**	**0.00**	2.81	**0.59**	0.01	2.93	**0.78**	**0.01**
	RCME.b.3	0.17	**0.00**	**0.00**	2.81	0.64	0.01	2.93	0.82	0.03

Table 1. Argoverse dataset planning evaluation. Note that the **.1** and **.3** variations using the same models/samples. In **.1** we selected the trajectory with highest probability/lowest cost. In **.3** we chose 3. Therefore, the minADE is the same for these variations. Variants of our method (gray) outperformed other algorithm in term of CR in all of the scenarios.

input sequence length τ and the prediction horizon T. In all cases, the data is partitioned into 20 sequences of frames. Thus, the time gap between two consecutive frames for $(\tau = 1, T = 1), (\tau = 2, T = 2), (\tau = 1, T = 3)$ are 0.1, 0.2 and 0.2 respectively.

We first evaluate the effectiveness of our approach in trajectory planning using a variety of metrics and compare our approach to multiple baselines. We quantitatively evaluate the performance of all these solutions in different planning horizons. In Section 4.3 we provide multiple ablation studies to show the effects of different modules and objective terms in the overall performance of the system.

4.1. Cost Map Estimation

Direct evaluation of predicted CMs is not straightforward as there is no groundtruth for them. We thus evaluate their quality by using them with the planning approach described in Section 3.3 and compare planning quality under the following metrics.

- **minADE:** We used the minimum average displacement error (minADE) $\min \frac{1}{T} \sum_{\tau}^{\tau+T} \|\hat{s} - s^*\|_2$ to measure the minimum drift of the trajectories generated by each model from the groundtruth. This metric is especially suitable for baselines producing multiple trajectories as well as the proposed method because it does not penalize the trajectories that are valid, but far from the groundtruth. For algorithms that only produce a single trajectory, minADE reduces to ADE. Since our solution aims at capturing the underlying reasons for the expert's behavior rather than merely generating trajectories, just minADE with respect to expert's trajectories is inadequate.

- **Potential Collision Rate (CR):** Each selected trajec-

Algorithms		$\tau = 1$ sec $T = 1$ sec			$\tau = 2$ sec $T = 2$ sec			$\tau = 1$ sec $T = 3$ sec		
Alg.	Aux	minADE	CR(%)	RV(%)	minADE	CR(%)	RV(%)	minADE	CR(%)	RV(%)
RCME.1	✓	0.18	0.00	0.00	2.45	0.67	0.01	2.92	0.84	0.01
RCME.1		0.22	0.00	0.00	2.76	0.89	0.01	3.18	4.02	0.01
RCME.3	✓	0.18	0.00	0.00	2.45	0.93	0.01	2.92	1.00	0.01
RCME.3		0.22	0.11	0.06	2.76	2.68	0.03	3.18	7.32	0.12

Table 2. CME with and without the auxiliary task

CME Methods	$\tau = 1$ sec $T = 1$ sec			$\tau = 2$ sec $T = 2$ sec			$\tau = 1$ sec $T = 3$ sec		
	minADE	CR(%)	RV(%)	minADE	CR(%)	RV(%)	minADE	CR(%)	RV(%)
RCME, with pred	**0.15**	**0.01**	**0.00**	**1.94**	**0.92**	0.01	**3.52**	**1.68**	0.05
RCME, without pred	0.34	0.79	**0.00**	3.13	7.18	**0.00**	3.99	11.14	**0.00**

Table 3. CME with and without the OGM prediction

tory is mapped to the future frames to check if it collides with any object in the scene. While the ego vehicle's behavior affects other cars' trajectories in the real world, for short horizons considered here we may ignore such interaction.

- **Road Violation (RV):** The selected trajectory is mapped to a drivable area to check for possible violations of traffic rules.

We compare our solution with the following baselines:

- **Behavior Cloning(BC):** We implemented a BC learner that receives a sequence of OGMs and the past trajectory for τ timesteps and generates trajectories close to those of the human driver. For fair comparison we use the same OGM predictor in our model. Trajectories are generated with one of two architectures, where BC-MLP uses four CNN layers with [32, 64, 64, 128] filters followed by three mlp layers with [64, 32, 2T] units and BC-LSTM uses a CNN encoder with [16, 32] filters to encode map and predicted ($\hat{O}$) or observed (O^*) OGMs and predict S for T timesteps.

- **Rule-Based Cost Map (RuleCM):** Instead of predicting CMs we use hard rules to shape a CM. We assign high cost to non-drivable areas and the occupied cells at the present time. The same trajectory generator as in Section 3.3 is used and driving trajectory with lowest cost is selected. This baseline highlights the importance of the *predicted* cost for motion planning.

- **Estimating Social-forecast Probability (ESP):** We compare our results to ESP [21] for the single agent, using the code published at https://github.com/nrhine1/precog. We did not do hyperparameter tuning, but we used both forward KL and symmetric cross entropy for the objective function and reported the best results. We sample 6 trajectories ($K = 6$) for evaluation. For CR and RV we chose top-1 and top-3 trajectories according to the model-assigned probability and reported the results for both. These variations are referred to as ESP.1 and ESP.3. Note that because we use the same samples from the same model to study if all of the generated trajectories are useful for planning, the minADE is the same for these variations.

- **Multiple Future Prediction (MFP):** MFP [28] is a multi-modal trajectory prediction solution. We use code from https://github.com/apple/ml-multiple-futures-prediction with adaptations to make it work for a single agent. We acknowledge that this adaptation affects the performance of MFP as the other agents' trajectories are the key inputs to this algorithm. In a complete AD system, such data may come from perception modules that detect and track other agents. Since we are studying the performance of prediction in the absence of such modules, we choose to test MFP in a limit case. We used 3 modes for MFP. Similar to ESP, we use two variations of sampled trajectories, where MFP.1 and MFP.3 refer to top-1 and top-3 trajectory selections respectively.

For our solution, we study the performance of both the RCME and the MSCME architectures. In one setting (variation **.a**) we use the trajectories generated according to Section 3.3, and in another setting (variation **.b**) we add the trajectories generated by the imitation network to the samples. Similar to ESP and MFP, we use top-1 and top-3 trajectories to evaluate the quality of the trajectories.

4.2. Planning Results

As shown in Table 1, all solutions have low collision rate (CR) for shorter horizons. As horizon gets longer, history shorter, and frequency lower, CR increases markedly for all the algorithms except for ours.

Both BC baselines have low ADE for shorter horizons ($T = 1, 2$). This is expected as they explicitly minimize the difference between predicted and expert's trajectories. But for the more challenging settings where they plan trajectories for 3 seconds based on 1 second of history, the generated trajectories have higher ADE, CR and RV. In contrast, even though our solution uses a trajectory sampler to propose trajectories it has low ADE, CR and RV in all settings. Adding the predicted trajectory by the imitation network to the samples helps with the performance in some scenarios.

Algorithms	$\tau = 1$ sec $T = 1$ sec			$\tau = 2$ sec $T = 2$ sec			$\tau = 1$ sec $T = 3$ sec		
	TP	TN	S_{100}	TP	TN	S_{100}	TP	TN	S_{100}
Diff. Learn	81.92	98.32	96.32	80.07	**99.08**	**96.52**	78.64	**99.26**	**97.33**
RCME	**82.13**	**98.31**	**97.48**	**81.96**	97.08	93.84	**81.38**	98.19	95.39

Table 4. OGM prediction with and without CME

RuleCM has a low RV percentage as we manually assign high values to the non-drivable grids. However, its high CR shows that it cannot handle dynamic objects well. This highlights the importance of the predictive nature of our proposed solution.

As mentioned above, we did not tune the hyperparameters for MFP and ESP on Argoverse. [18] reports better performance for these algorithms but we could not replicate those results. Multiple factors including the difference in hyperparameters, preprocessing of the data, prediction horizon and number of samples may have contributed to this performance gap. We also use the single-agent variant for both, which forces these algorithms to capture interactions using high dimensional inputs only. This can potentially lead to a decline in performance. The high CR of ESP suggests that its architecture may not be effective in capturing dynamics of the environment and interactions from the high dimensional features. Moreover, the increase in CR and RV for top-3 trajectories over top-1 trajectories and also higher CR and RV even when the performance of these algorithms are close to the proposed method in terms of minADE show that the multi-modality of these algorithms may not be directly suitable for planning. In other words, a small minADE is not an indication of the admissibility of all the samples. The authors of ESP used a similar architecture in [20] and [22] for planning; the authors of MFP also did a brief study on using MFP for planning [28]. Given the success of these algorithms in multi-modal trajectory prediction, our experiments suggest that in order to assess their potential in motion planning they should also be evaluated with planning metrics such as CR on real-world data.

4.3. Ablation Study

4.3.1 Objective Function

To study the contribution of the proposed auxiliary task we trained two models with or without it and summarized the results from the RCME.a setting in Table 2. The results show that for the more challenging scenario (τ = 3, T = 1) CR is lower when the auxiliary objective is used. Intuitively, the auxiliary task does not affect RV as much because $\mathcal{L}_p$ takes the static environment into account. But as the dynamics of the environment gets more complex, the role of the auxiliary objective gets more clear.

For the settings with top-3 trajectories, we explicitly chose trajectories from different clusters so as to examine the ability of the system to reason about different scenarios. This leads to a larger effect of the auxiliary objective (RCME.3 rows in Table 2) even in the easier scenarios, suggesting that the auxiliary task contributes to better generalization.

Discussion: We tried to employ the planning objective introduced in [35] to compare the results. However, the network failed to estimate CMs. We believe due to the highly sparse nature of that objective, the perception modules in the architecture are crucial to lead the training.

4.3.2 Network Architecture

We also conducted ablation studies on different modules in our architecture. First, we replace the OGM predictor in the MSCME architecture with a CNN encoder so that the model directly estimates the CM without the help from OGM predictions. We do this only to MSCME, because in RCME the CM estimation is embedded inside the OGM prediction system. The results are summarized in Table 4. This change leads to a large performance gap in terms of CR, suggesting that simultaneous OGM prediction extracts better mid-level features suitable for reasoning about environment dynamics.

While the quality of OGM prediction is not the focus of this paper, studying it offers more insight into the overall performance of the system. Thus, we compared the OGM predictions of **RCME** with the **Difference Learning Architecture** in [16]. The metrics we use are percentage of True Positive (TP), True Negative (TN). We also multiply SSIM by 100 (S_{100}) to make it the same scale as the other metrics. The results are summarized in Table 3. It is not surprising that RCME has higher TP rate compared to Difference Learning, because the additional CM estimation task brings more information to the system, leading to more accurate predictions.

5. Conclusion

The definition of driving cost is highly ambiguous and should encode human's driving behavior as well as the environment characteristic such as road structure. In this work, we proposed a novel fully self-supervised approach for estimating high dimensional CMs. Due to the importance of the prediction in CME, part of the network is dedicated for predicting high-dimensional OGMs. Input and predicted OGMs, contextual information as well as the human driving behavior are then utilized to extract features required to encode expert's demonstration. We applied the proposed method to Argoverse dataset and illustrated the effectiveness of our approach in different planning horizons.

References

[1] 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.

[2] 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.

[3] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[4] Ming-Fang Chang, John W Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3d tracking and forecasting with rich maps. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[5] Lu Chi and Yadong Mu. Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *arXiv preprint arXiv:1708.03798*, 2017.

[6] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, Sebastian Thrun, and Ronald C Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[7] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[8] Julie Dequaire, Peter Ondrúška, Dushyant Rao, Dominic Wang, and Ingmar Posner. Deep tracking in the wild: End-to-end tracking using recurrent neural networks. *The International Journal of Robotics Research*, 37(4-5):492–512, 2018.

[9] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016.

[10] Stefan Hoermann, Martin Bach, and Klaus Dietmayer. Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2056–2063. IEEE, 2018.

[11] Xuemin Hu, Long Chen, Bo Tang, Dongpu Cao, and Haibo He. Dynamic path planning for autonomous driving on various roads with avoidance of static and moving obstacles. *Mechanical Systems and Signal Processing*, 100:482–500, 2018.

[12] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE, 2017.

[13] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[14] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 3569–3577, 2018.

[15] Isaac Miller and Mark Campbell. A mixture-model based algorithm for real-time terrain estimation. *Journal of Field Robotics*, 23(9):755–775, 2006.

[16] Nima Mohajerin and Mohsen Rohani. Multi-step prediction of occupancy grid maps with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10600–10608, 2019.

[17] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[18] Seong Hyeon Park, Gyubok Lee, Jimin Seo, Manoj Bhat, Minseok Kang, Jonathan Francis, Ashwin Jadhav, Paul Pu Liang, and Louis-Philippe Morency. Diverse and admissible trajectory forecasting through multimodal context understanding. In *European Conference on Computer Vision*, pages 282–298. Springer, 2020.

[19] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6059–6066. IEEE, 2017.

[20] Nicholas Rhinehart, Kris M Kitani, and Paul Vernaza. R2p2: A reparameterized pushforward policy for diverse, precise generative path forecasting. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 772–788, 2018.

[21] Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2821–2830, 2019.

[22] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*, 2018.

[23] Sascha Rosbach, Vinit James, Simon Großjohann, Silviu Homoceanu, and Stefan Roth. Driving with style: Inverse reinforcement learning in general-purpose planning for automated driving. *arXiv preprint arXiv:1905.00229*, 2019.

[24] Amir Sadeghian, Ferdinand Legros, Maxime Voisin, Ricky Vesel, Alexandre Alahi, and Silvio Savarese. Car-net: Clairvoyant attentive recurrent network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 151–167, 2018.

[25] Julian Schlechtriemen, Kim Peter Wabersich, and Klaus-Dieter Kuhnert. Wiggling through complex traffic: Planning trajectories constrained by predictions. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1293–1300. IEEE, 2016.

[26] Shai Shalev-Shwartz, Nir Ben-Zrihem, Aviad Cohen, and Amnon Shashua. Long-term planning by short-term prediction. *arXiv preprint arXiv:1602.01580*, 2016.

[27] Dong Hun Shin, Sanjiv Singh, and W Whittaker. Path generation for a robot vehicle using composite clothoid segments. *IFAC Proceedings Volumes*, 25(6):443–448, 1992.

[28] Charlie Tang and Russ R Salakhutdinov. Multiple futures prediction. In *Advances in Neural Information Processing Systems*, pages 15424–15434, 2019.

[29] Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.

[30] Christopher Urmson, Joshua Anhalt, Hong Bae, J. Andrew (Drew) Bagnell, Christopher R. Baker, Robert E. Bittner, Thomas Brown, M. N. Clark, Michael Darms, Daniel Demitrish, John M. Dolan, David Duggins, David Ferguson, Tugrul Galatali, Christopher M. Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Maxim Likhachev, Bakhtiar Litkouhi, Alonzo Kelly, Matthew McNaughton, Nick Miller, Jim Nickolaou, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Varsha Sadekar, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod M. Snider, Joshua C. Struble, Anthony (Tony) Stentz, Michael Taylor, William (Red) L. Whittaker, Ziv Wolkowicki, Wende Zhang, and Jason Ziglar. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, 25(8):425–466, June 2008.

[31] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[32] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993. IEEE, 2010.

[33] Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2089–2095. IEEE, 2016.

[34] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182, 2017.

[35] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019.

[36] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.