SEPARATING THE WORLD AND EGO MODELS FOR SELF-DRIVING

Vlad Sobal[1], Alfredo Canziani[2], Nicolas Carion[2], Kyunghyun Cho[1, 2, 3, 4], Yann LeCun[1, 2, 5]
[1] Center for Data Science, New York University
[2] Courant Institute, New York University
[3] Prescient Design, Genentech
[4] CIFAR Fellow
[5] Meta AI Research
us441@nyu.edu

ABSTRACT

Training self-driving systems to be robust to the long-tail of driving scenarios is a critical problem. Model-based approaches leverage simulation to emulate a wide range of scenarios without putting users at risk in the real world. One promising path to faithful simulation is to train a forward model of the world to predict the future states of both the environment and the ego-vehicle given past states and a sequence of actions. In this paper, we argue that it is beneficial to model the state of the ego-vehicle, which often has simple, predictable and deterministic behavior, separately from the rest of the environment, which is much more complex and highly multimodal. We propose to model the ego-vehicle using a simple and differentiable kinematic model, while training a stochastic convolutional forward model on raster representations of the state to predict the behavior of the rest of the environment. We explore several configurations of such decoupled models, and evaluate their performance both with Model Predictive Control (MPC) and direct policy learning. We test our methods on the task of highway driving and demonstrate lower crash rates and better stability. The code is available at https://github.com/vladisai/pytorch-PPUU/tree/ICLR2022.

1 INTRODUCTION

Models of the world have proven to be useful for various tasks (Hafner et al., 2020; Ebert et al., 2018; Kaiser et al., 2020), including self-driving (Henaff et al., 2019; Ha & Schmidhuber, 2018).

In their work, Henaff et al. (2019) develop a model-based approach to policy learning for highway driving. The world model in the proposed system is trained to predict semantic rasterization of the top-down view of a section of a highway around the ego-vehicle, as well as the position and velocity of the ego-vehicle. The policy model interacts with this world model and gets updated by following the gradient calculated by backpropagation from a handcrafted cost through the world model and into the policy parameters. This approach allows for training policies without needing additional on-policy data, which is a very important advantage for self-driving.

However, the proposed world model has a few important limitations. The world model needs to predict both the image of the top-down view and the vehicle state. The two tasks are very different: ego-vehicle state prediction is deterministic and can be computed using a few kinematic equations, while the environment has to be represented by a more complex model capable of representing multimodal predictions to capture the variety of behaviors of other traffic participants. This difference suggests the need for two distinct models, not one. Another limitation of the method proposed by Henaff et al. (2019) is the cost function that is non-differentiable with respect to the position of the vehicle. The gradients flow only through the prediction of the top-down image while omitting important learning signal that can be obtained from the position of the car.

In the present work, we claim that problems that require modeling of agents' motions in a complex stochastic environment should be addressed with two distinct models: a simple and deterministic

kinematic model that predicts ego-agents' state in the environment, and a complex model of the environment that is capable of addressing the problem's stochasticity and multimodality. Separating these models provides several advantages. First, we can leverage the knowledge of kinematics to build an exact ego model. Second, when the context allows, we can even make the two models completely independent of each other, making model-predictive control simpler and more efficient.

We build on top of the work of Henaff et al. (2019) and make the following contributions:

- we introduce a way to split a world model for self-driving into an environment model and an ego model, and compare different ways of integrating them into one system;
- we propose a novel approach that makes the two models independent. We demonstrate that such an approach is faster in certain settings while achieving better performance on the task of highway driving;
- we design a cost function that is differentiable with respect to the position and velocity of the ego vehicle;

2 PROBLEM DESCRIPTION

We consider the problem of autonomous highway driving in this paper, although our approach is applicable to controlling any (similar) autonomous system. We consider three variables at each time step t. They are the self-state $\boldsymbol{s}_t^{\text{self}} \in \mathbb{S}^{\text{self}}$, the self-action $\boldsymbol{a}_t^{\text{self}} \in \mathbb{A}^{\text{self}}$ and the environment state $\boldsymbol{s}_t^{\text{env}} \in \mathbb{S}^{\text{env}}$. As the names suggest, the first two are the state of and action taken by the autonomous vehicle under control (ego-car), while the environment state includes everything except the ego-car, such as other cars and any other objects and agents. In our setup $\boldsymbol{s}_t^{\text{self}} \in \mathbb{R}^5, \boldsymbol{s}_t^{\text{self}} = (x_t, y_t, u_t^x, u_t^y, s_t)$, where (x_t, y_t) are coordinates of the center of the rear axle (approximated as the center of the rear end of the car), (u_t^x, u_t^y) is a unit direction vector, and s_t is a scalar denoting the speed. $\boldsymbol{a}_t^{\text{self}} \in \mathbb{R}^2$ is the action taken by the controlled agent at time t, with $a_{t,0}$ and $a_{t,1}$ denoting the applied acceleration and rotation strength respectively. $\boldsymbol{s}_t^{\text{env}}$ is a rasterized mid-level representation of a portion of the highway around the ego-vehicle. An example of such representation is provided in the top row of figure 2. We also assume the availability of a cost function $C : \mathbb{S}^{\text{env}} \times \mathbb{S}^{\text{self}} \times \mathbb{A}^{\text{self}} \to \mathbb{R}^+$ that, given states and actions at step t, calculates the cost. In this work, C has been handcrafted and includes components to account for proximity of other vehicles, driving off the road and closeness to the lane center. Having all these components, the goal is then to find a policy π that minimizes the cumulative cost $J(\pi) = \mathbb{E}_{(\boldsymbol{s}_1^{\text{env}},\ldots,\boldsymbol{s}_T^{\text{env}}),(\boldsymbol{s}_1^{\text{self}},\ldots,\boldsymbol{s}_T^{\text{self}})\sim\pi}\left[\sum_{t=1}^{T} C\left(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_t^{\text{self}}, \pi(\boldsymbol{s}_{t-1}^{\text{env}}, \boldsymbol{s}_{t-1}^{\text{self}})\right)\right]$, where T is episode length.

Dependencies To create a world model for this problem, we must consider the dependencies among these state and action variables. We start by exhaustively enumerating these dependencies. State components depend on the state information at the previous time step and the action, while the action depends on the state information:

- $\boldsymbol{s}_{t+1}^{\text{self}} \leftarrow \boldsymbol{a}_t^{\text{self}}, \boldsymbol{s}_t^{\text{self}}, \boldsymbol{s}_t^{\text{env}}$
- $\boldsymbol{s}_{t+1}^{\text{env}} \leftarrow \boldsymbol{a}_t^{\text{self}}, \boldsymbol{s}_t^{\text{self}}, \boldsymbol{s}_t^{\text{env}}$
- $\boldsymbol{a}_t^{\text{self}} \leftarrow \boldsymbol{s}_t^{\text{self}}, \boldsymbol{s}_t^{\text{env}}$

The goal of creating a world model then boils down to building a function approximator that predicts $\boldsymbol{s}_{t+1}^{\text{self}}$ and $\boldsymbol{s}_{t+1}^{\text{env}}$ variables given their dependencies. Once we have the models for the states $(\boldsymbol{s}_t^{\text{self}}, \boldsymbol{s}_t^{\text{env}})$, we can minimize the cost, or the sum of it over time, w.r.t. the action sequence $(\boldsymbol{a}_1^{\text{self}}, \ldots, \boldsymbol{a}_T^{\text{self}})$, on which we can train a policy π for driving the autonomous vehicle.

3 WORLD MODELING

In this section, we present the kinematics model and three possible ways of integrating it with the environment model. In section 4 we present two ways of obtaining a policy using the world

model. We then choose 4 combinations of environment model and policy learning set-ups and show experimental results in section 5. The 4 combinations we use are shown in figure 1. We review related work in section 6, and conclude with section 7.

Kinematics model We propose to simplify the dependency pattern of $\boldsymbol{s}_t^{\text{self}}$: $\boldsymbol{s}_{t+1}^{\text{self}} \leftarrow \boldsymbol{a}_t^{\text{self}}, \boldsymbol{s}_t^{\text{self}}, \cancel{\boldsymbol{s}_t^{\text{env}}}$. If we assume that the state of the environment $\boldsymbol{s}_t^{\text{env}}$ does not affect the ego vehicle's state $\boldsymbol{s}_{t+1}^{\text{self}}$, we can resort to a simple bicycle kinematics model for state prediction. This assumption holds unless there is a collision between the ego car and an object in the environment.

We use the following formulation of the kinematic bicycle model:

$$x_{t+1} = x_t + s_t u_t^x \Delta t \tag{1}$$
$$y_{t+1} = y_t + s_t u_t^y \Delta t \tag{2}$$
$$s_{t+1} = s_t + a_{t,0} \Delta t \tag{3}$$
$$(u_{t+1}^x, u_{t+1}^y) = \text{unit}[(u_t^x, u_t^y) + a_{t,1} \Delta t (u_t^y, -u_t^x)] \tag{4}$$

Where Δt is the time step (we use $\Delta t = 0.1\,\text{s}$), $\text{unit}(\boldsymbol{v}) = \frac{\boldsymbol{v}}{|\boldsymbol{v}|}$. Equation 4 can be intuitively understood as adding to the current direction vector an orthogonal unit vector multiplied by the turning command and the time step.

Environment model The environment model f_θ^{env} is directly inspired by the work of Henaff et al. (2019). We use the same architecture in all our experiments with slight changes to the input. In all cases, the model f_θ^{env} takes as input $\boldsymbol{s}_t^{\text{env}}$ and outputs $\boldsymbol{s}_{t+1}^{\text{env}}$. Depending on the set-up, the f_θ^{env} may have other inputs and/or outputs. An example of the prediction of a sequence $\boldsymbol{s}_t^{\text{env}}$ is shown in the top row of figure 2. To find the best way of integrating the kinematic model into the system, we experiment with three configurations of the environment model:

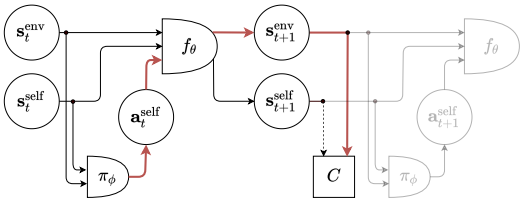
1. Coupled Forward Model (CFM) is directly taken from Henaff et al. (2019). In this configuration $f_\theta^{\text{env}} : \mathbb{S}^{\text{env}} \times \mathbb{S}^{\text{self}} \times \mathbb{A}^{\text{self}} \to \mathbb{S}^{\text{env}} \times \mathbb{S}^{\text{self}}$, $(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_t^{\text{self}}, \boldsymbol{a}_t^{\text{self}}) \mapsto (\boldsymbol{s}_{t+1}^{\text{env}}, \boldsymbol{s}_{t+1}^{\text{self}})$. There is no explicit f^{self} model, instead the world model f_θ^{env} is trained to predict both $\boldsymbol{s}_{t+1}^{\text{env}}$ and $\boldsymbol{s}_{t+1}^{\text{self}}$. The diagram of a set-up using this model is shown in figure 1a. This approach models all dependencies described in section 2 with a single model f_θ^{env}.

2. Coupled Forward Model with Kinematics (CFM-KM) extends the CFM model, but utilizes the proposed kinematic model and the associated independence assumption to better model $\boldsymbol{s}_t^{\text{self}}$. We have $f_\theta^{\text{env}} : \mathbb{S}^{\text{env}} \times \mathbb{S}^{\text{self}} \to \mathbb{S}^{\text{env}}$, $(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_{t+1}^{\text{self}}) \mapsto \boldsymbol{s}_{t+1}^{\text{env}}$. The model of the environment f_θ^{env}, instead of taking $\boldsymbol{a}_t^{\text{self}}$ as input, takes the prediction of $\boldsymbol{s}_{t+1}^{\text{self}}$ provided by $f^{\text{self}}(\boldsymbol{s}_t^{\text{self}}, \boldsymbol{a}_t^{\text{self}})$. Thus, f_θ^{env} does not need to learn the kinematics. Diagrams of two methods using such set-up are shown in figures 1b and 1c.

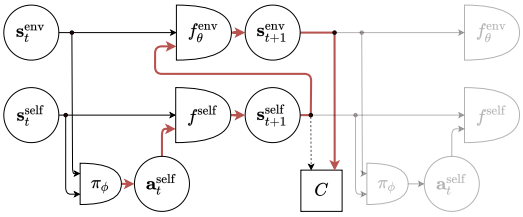
3. Decoupled Forward Model with Kinematics (DFM-KM) In this case, to make the approximation more efficient, we introduce another change to the dependency pattern: $\boldsymbol{s}_{t+1}^{\text{env}} \leftarrow \cancel{\boldsymbol{a}_t^{\text{self}}}, \cancel{\boldsymbol{s}_t^{\text{self}}}, \boldsymbol{s}_t^{\text{env}}$. This severs the dependency between the environment and the state of the ego-vehicle. Now, we can run the environment model f^{env} separately from f^{self}. We then have $f_\theta^{\text{env}} : \mathbb{S}^{\text{env}} \to \mathbb{S}^{\text{env}}$, $\boldsymbol{s}_t^{\text{env}} \mapsto \boldsymbol{s}_{t+1}^{\text{env}}$. As before, $\boldsymbol{s}_{t+1}^{\text{self}}$ is predicted using $f^{\text{self}}(\boldsymbol{s}_t^{\text{self}}, \boldsymbol{a}_t^{\text{self}})$. A diagram of a set-up using such pattern is shown in figure 1d.

4 Policy

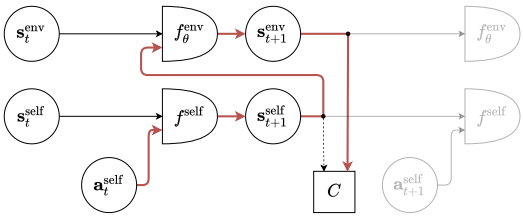
Cost function CFM PL uses the same cost function as Henaff et al. (2019). CFM-KM and CFM-KM MPC add one modification: the off-road component. Adding that cost component to CFM PL cost function does not change performance by much, we show results in appendix C. The resulting cost function $C(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_t^{\text{self}})$ contains components to account for proximity to other road users, crossing lane demarcations and driving off the road. For the DFM-KM set-up, we implement C^{km}, a modification of the original cost function C that is differentiable with respect to $\boldsymbol{s}^{\text{self}}$. In its original version, the cost is used for backpropagation through $\boldsymbol{s}_t^{\text{env}}$ and the forward model f_θ^{env} (see figure 1a). With the decoupled model, we no longer require the forward model to be differentiable and instead perform backpropagation through $\boldsymbol{s}^{\text{self}}$. To calculate the cost of taking a sequence of actions



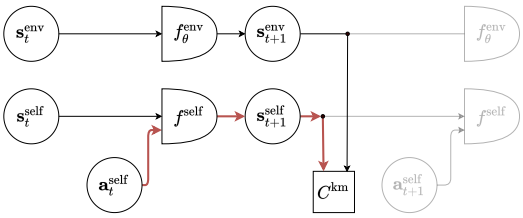
(a) **CFM Policy** method as proposed by Henaff et al. (2019). There's no kinematics model f^{self}. Instead, the forward model f_θ^{env} is tasked with learning the kinematics as well as the predicting the trajectories of other agents.



(b) **CFM-KM Policy.** Here, the forward model does not predict $\boldsymbol{s}_{t+1}^{\text{self}}$. The exact kinematics model f^{self} predicts it instead and passes to f_θ^{env} as input.



(c) **CFM-KM MPC.** Like in CFM-KM Policy, we use the kinematics model f^{self} to help f^{env}. Additionally, instead of learning a policy, we minimize C by directly optimizing the action $\boldsymbol{a}_t^{\text{self}}$ with a few steps of gradient descent.



(d) **DFM-KM MPC.** Here we assume independence of f_θ^{env} from $\boldsymbol{s}_t^{\text{self}}$. The gradients are only propagated through f^{self}. We do not require f_θ^{env} to be differentiable.

Figure 1: **Diagrams of the compared methods** The circles represent values, while the half ellipses represent functions. Arrows represent information flow, and the red color denotes the flow of the gradients. Grayed out areas depict the flow into the prediction of next time step. Dashed lines depict non-differentiable paths. In, CFM-based methods, the cost C is not differentiable w.r.t. $\boldsymbol{s}^{\text{self}}$.

of length T $(\boldsymbol{a}_1^{\text{self}}, \dots, \boldsymbol{a}_T^{\text{self}})$, we first run f_θ^{env} to obtain the predictions of $(\boldsymbol{s}_1^{\text{env}}, \dots, \boldsymbol{s}_T^{\text{env}})$ (see the top row of figure 2). Then, we use f^{self} to predict $(\boldsymbol{s}_1^{\text{self}}, \dots, \boldsymbol{s}_T^{\text{self}})$, which are then used to create masks shifted to locations that match the sequence of $\boldsymbol{s}^{\text{self}}$ (see the bottom row of figure 2). The mask is shifted in a differentiable manner to allow gradient propagation to $\boldsymbol{s}^{\text{self}}$. The masks are then multiplied with individual channels of the predicted $\boldsymbol{s}^{\text{env}}$ to obtain different components of the cost, which are then combined into one scalar with corresponding weighting coefficients. For a more in-detail explanation of the cost calculation, see appendix D. With this method, we can backpropagate into the action sequence $(\boldsymbol{a}_1^{\text{self}}, \dots, \boldsymbol{a}_T^{\text{self}})$, update it following the negative direction of the gradients, and repeat the whole process until the cost is low enough. Note that since we do not backpropagate through $\boldsymbol{s}_t^{\text{env}}$ or f_θ^{env}, we do not need to re-run the forward model at each optimization step.

Policy We utilize two approaches for obtaining the driving policy.

1. Model Predictive Control (MPC) At step $t = 0$, having a sequence of planned actions of length T $(\boldsymbol{a}_0^{\text{self}}, \dots, \boldsymbol{a}_{T-1}^{\text{self}})$, we want to minimize the cost associated with that plan. We first use forward models f_θ^{env} and f^{self} to predict $(\boldsymbol{s}_1^{\text{env}}, \dots, \boldsymbol{s}_T^{\text{env}})$ and $(\boldsymbol{s}_1^{\text{self}}, \dots, \boldsymbol{s}_T^{\text{self}})$, and then use the cost C to obtain the total cost of the predicted trajectory $J = \sum_{t=1}^{T} C(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_t^{\text{self}}, \boldsymbol{a}_{t-1}^{\text{self}}) \cdot \gamma^t$, where γ is the discounting factor that is set to 0.99. Assuming that the cost C calculation is differentiable w.r.t. the actions, we can backpropagate the gradients into the sequence of actions (see figure 1c and 1d). We then do several steps of gradient descent to update the action sequence to minimize the cost. Having the optimized sequence of actions, we then take the first action $\boldsymbol{a}_1^{\text{self}}$ in the sequence and discard the rest, only to re-plan again at the next time step. For more details, see the pseudocode of DFM-KM MPC and CFM-KM MPC in figure 4 in the appendix.

2. Policy Learning (PL) This approach is inspired by the method proposed by Henaff et al. (2019). $\boldsymbol{a}_t^{\text{self}}$ can be modeled using a model of the policy $\pi_\phi(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_t^{\text{self}})$. We train a policy to

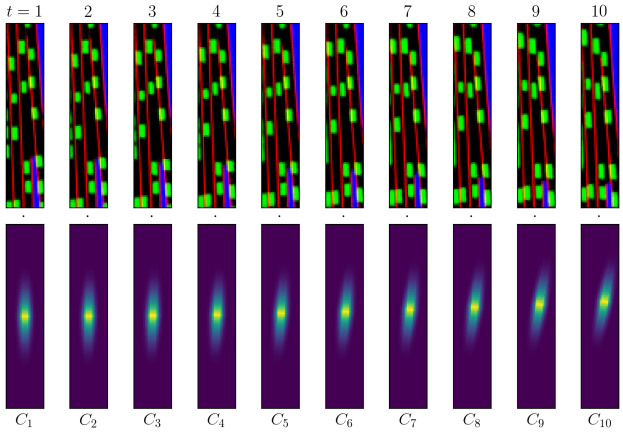


Figure 2: **Cost calculation.** The top row shows f_θ^{env} predictions for $\boldsymbol{s}^{\text{env}}$ across the time steps. Red, green, and blue denote lane demarcations, road users, and off-road regions respectively. The bottom row shows masks used for calculating cost components. The masks' location is differentiably adjusted based on f^{self} prediction of $\boldsymbol{s}_{1:10}^{\text{self}}$. $\cdot$ represents dot product. The bottom row masks are multiplied with individual channels of the corresponding top row images to obtain the values of $C_{1:10}$.

minimize the sum of the costs $J = \sum_{t=1}^{T} C(\boldsymbol{s}_t^{\text{env}}, \boldsymbol{s}_t^{\text{self}}, \pi_\phi(\boldsymbol{s}_{t-1}^{\text{env}}, \boldsymbol{s}_{t-1}^{\text{self}})) \cdot \gamma^t$ over a roll-out trajectory of T steps. This can be done by computing the negative gradient of the sum of the costs w.r.t. the policy's parameters ϕ and repeatedly taking the step towards it (see figure 1a).

In the case of both MPC and PL, the proposed sparse dependency pattern simplifies backpropagation. Gradients flow only through the self-states $\boldsymbol{s}_t^{\text{self}}$, as we have severed the dependency between the environment state and the self-state. This dramatically lowers the number of interactions involved in the backward pass. This should alleviate the issue of vanishing gradients and improve the quality of the gradients flowing to actions $\boldsymbol{a}_t^{\text{self}}$.

Expected benefits We expect the methods proposed in section 3 to improve the obtained policies in the following ways:

1. We expect the policies to have improved generalization and lower variance of predicted actions when using the kinematics model and/or decoupled forward model. Backpropagating through a simpler and more exact model should enable us to train a more robust policy.
2. Decoupled forward model simplifies the model that connects the action to the cost. Therefore, we expect each backward pass to take less time, making MPC faster than in the coupled approach.

We test both the existence and degree of these benefits in our experiments.

A potential drawback The obvious drawback of the sparsification in the proposed DFM-KM is that the environment state and self-state may eventually become incompatible with each other. Because objects in the environment are not aware of the ego-car, some of them may eventually overlap with the ego-car, resulting in an unrealistic situation, such as squeezing in a traffic jam. To avoid such unrealistic situations from impacting the policy, we only use a limited roll-out when using the proposed sparse dependency pattern. We argue that this is fine from two perspectives. First, even with the conventional dense dependency pattern, learning a policy by backpropagating through a recurrent network, which is how a world model is often implemented, is challenging because of vanishing or exploding gradients. Therefore, the effective horizon of backpropagation does not decrease much by using a limited roll-out with the sparse dependency pattern. Second, our task of lane following does not require long-range planning by construction. The policy only needs to repeat short-term goals, *i.e.* to maintain speed and distance from other objects on the road, over and over.

Table 1: **Crash rates comparison.** We compare episode failure rates in two simulation setups: replay, where other cars are following the trajectories from the recorded dataset; and interactive simulation, where other cars are controlled either by CFM-KM PL or CFM PL methods. Lower crash rates are better.

		Interaction Policy	
Method	**Fixed Replay**	**CFM-KM PL**	**CFM PL**
CFM PL	25.2 ± 3.0	9.5 ± 1.0	5.7 ± 2.6
CFM-KM PL	15.1 ± 1.9	$\mathbf{1.0 \pm 0.1}$	$\mathbf{1.1 \pm 0.3}$
CFM-KM MPC	25.4 ± 1.4	4.2 ± 1.1	3.3 ± 0.9
DFM-KM MPC	$\mathbf{13.2 \pm 1.2}$	1.5 ± 0.5	1.7 ± 0.6

5 EXPERIMENTS

Dataset We test our methods on the task of highway driving on the NGSIM I-80 dataset (Halkias & Colyar, 2006). The dataset consists of highway driving scenarios recorded from multiple cameras mounted above a section of Highway I-80 in California. The recordings take place at different times of day to maximize the diversity of traffic densities. We follow the pre-processing steps of Henaff et al. (2019) and obtain cars' dimensions and trajectories on the highway. We use the same dataset split of 80%, 10%, 10% for training, validation, and testing respectively.

Crash rates comparison We compare the selected combinations of approaches to policy learning and forward modeling proposed in section 3. The components used by the methods are encoded in the names. For implementation details, refer to appendix B. The compared methods are:

(a) CFM PL See figure 1a. This is the approach proposed by Henaff et al. (2019).

(b) CFM-KM PL See figure 1b. This augments CFM PL by adding the exact kinematic model following the method described in section 3.

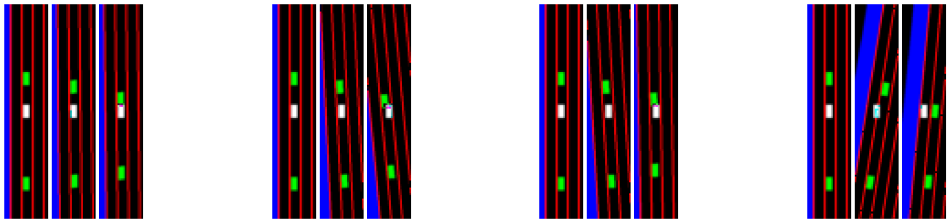
(c) CFM-KM MPC See figure 1c. Same as CFM-KM PL, but it uses MPC to find the best action.

(d) DFM-KM MPC See figure 1d. As described in section 3, this combines MPC with exact kinematic model, decoupled forward model, and the modified cost function.

We test all methods in two settings: replay simulation and interactive simulation. Replay simulation simply replays the trajectories of all cars except one, which is controlled by the method we are evaluating. This is the same evaluation protocol as was used by Henaff et al. (2019). This evaluation method has a severe limitation: other cars' actions are simply replayed from the dataset, and therefore are independent of the ego car's actions. Some unrealistic situations may happen, for example, the ego-car can be squeezed by other cars in a traffic jam if the ego car picks a different trajectory from the one that the original vehicle followed during data recording. Since the proposed DFM-KM MPC method assumes exactly such independence, replay evaluation results may be biased in its favor. To address this problem, we also show the results of interactive simulation. Inspired by (Bergamini et al., 2021), we implement interactive simulation by controlling the ego-car with the selected method and controlling all other cars with either the CFM-KM PL or the CFM PL. We do not experiment with controlling other cars with MPC because it is orders of magnitude slower than doing one forward pass with a policy model (see table 2), rendering such evaluation setup too slow to be practical. The results are shown in table 1. DFM-KM MPC that uses the decoupled forward model with kinematics achieves the best performance in replay setting, closely trailed by the CFM-KM PL policy learned with the enhanced coupled forward model. This suggests that augmenting the forward model with exact kinematics equations gives a great boost in performance compared to the model that has to learn the kinematics from data. In the interactive setting, the CFM-KM PL performs slightly better, meaning that the independence assumption indeed biases the results somewhat in the replay evaluation in favor of DFM-KM MPC. However, DFM-KM MPC outperforms CFM-KM MPC by a big margin in all settings, showing that propagating the gradients through $\boldsymbol{s}^{\text{self}}$ and decoupling the forward model indeed improves the gradients' quality, allowing MPC to efficiently find better actions.

Table 2: **Time performance and output variance**. To measure agreement among policies using the same method but different seeds, we run the policies on the same input and calculate standard deviation of the produced actions.

		Standard deviation across seeds		
Method	**Milliseconds per simulation step**	**Acceleration**	**Turning**	**Average**
CFM PL	$\mathbf{1.2 \pm 0.0}$	1.08	1.00	1.04
CFM-KM PL	$\mathbf{1.2 \pm 0.0}$	1.07	**0.70**	0.88
CFM-KM MPC	1162.6 ± 34.9	1.05	0.77	0.91
DFM-KM MPC	509.5 ± 18.5	**0.78**	0.83	**0.80**



(a) **CFM PL** (b) **CFM-KM PL** (c) **CFM-KM MPC** (d) **DFM-KM MPC**

Figure 3: **Stress-testing the proposed methods.** The controlled car (white, in the center) is cruising between two other cars (green) when the car directly in front brakes suddenly. DFM-KM MPC manages to react in time, while other methods fail.

Time performance Another benefit of the decoupled approach is the improved speed performance of MPC. To demonstrate that, we measure the average time needed to evaluate one time step, and show the results in table 2. DFM-KM MPC needs about half the amount of time needed for the CFM-KM MPC. However, it is still orders of magnitude slower than running a trained policy.

Variance of actions We also test if the sparse dependency pattern facilitates more robust predictions by the policies. We show results in table 2. We observe that introducing the kinematic model helps to make the behavior more robust with respect to random initialization, with additional improvement gained from applying the proposed DFM-KM method. For an in-detail explanation of how these variances were computed, see appendix A.

Stress-testing We test the proposed methods on a hand-designed scenario — controlling an agent on a highway while cruising between two cars, when the car directly in the front brakes suddenly. We show the results in figure 3. We observe that only DFM-KM MPC method manages to successfully complete the scenario. CFM-KM MPC fails, showing that backpropagating through f_θ^{env} is not as efficient as backpropagating through f^{self} in DFM-KM MPC approach. We hypothesize that two factors help DFM-KM MPC here. First, in such extreme scenarios the decoupled model has an advantage since it is unreasonable to expect that the car in the front will react to the ego-agent. The independence assumption is justified, and helps the optimization process to find the best action. Second, a learned policy, trained to solve multiple different scenarios, likely becomes a smooth function that cannot take extreme values, while MPC is not restrained by model capacity and can find the minimum of the cost function C better in such extreme cases.

6 Related Work

Our work is related to the research on model-predictive control and motion planning methods. These are areas with decades of research; for comprehensive overviews of these topics we refer the reader to the books on optimal control (Bryson et al., 1979; Bertsekas, 2005), and motion plan-

ning (LaValle, 2006). In this section, we mainly focus on the recent methods that combine the existing approaches with deep neural networks.

Motion planning and Trajectory following Paden et al. (2016) provide a survey of the existing methods for self-driving. Approaches range from graph-search methods, such as A* (Ziegler et al., 2008), to dynamic programming (Montemerlo et al., 2008). In our work, trajectory following is simplified as we are acting in a simulator with a perfectly controllable car, and we focus on motion planning instead.

World modeling has proved to be a great approach to many tasks due to superior sample complexity in policy learning (Nagabandi et al., 2017), although it comes with some caveats, such as the danger of compounding errors (Asadi et al., 2019), and stochasticity (Denton & Fergus, 2018). World model used in our work combines ideas of action-conditioned model (Oh et al., 2015), and stochastic video prediction (Babaeizadeh et al., 2018). Such models have also been used in (Hafner et al., 2020) and (Henaff et al., 2017).

Model Predictive Control has been used widely for self-driving. Zhang et al. (2018) use MPC for planning and collision avoidance, Drews et al. (2018) use MPC to build an impressive system that drives a scaled-down vehicle at high speeds around a track. Our approach can be viewed as a type of Stochastic MPC (Heirung et al., 2018), or Scenario-based MPC (Schildbach et al., 2014; Cesari et al., 2017), where the trajectory is optimized for a limited number of future scenarios (in our case this number is 1, but it can easily be increased).

Kinematic models have been used extensively in self-driving applications. Often, cars are approximated by simplified models, such as unicycle (Kamenev et al., 2021) or bicycle models (Cesari et al., 2017). These models are particularly useful when adding inductive bias to models to produce realistic trajectories in path planning or behavior prediction (Salzmann et al., 2020). Kong et al. (2015) provide an overview of kinematics and dynamics models used for self-driving and apply them to path-following. The work of Scheel et al. (2021) is particularly close to ours as they also propose a differentiable kinematic model for training a self-driving policy. However, there is no trained environment model, and the predictions of the other road users are replaced with log replay.

Interactive simulation is a long-standing problem in self-driving cars development. Bergamini et al. (2021) proposed a method that uses GANs (Goodfellow et al., 2014) to generate the initial state, and then sequentially apply a learned policy to each of the generated agents. Suo et al. (2021) propose a system that models the agents' behavior jointly, making more consistent predictions.

7 Discussion & Conclusion

We presented a novel design of a world model for an agent with known kinematics in a complex stochastic environment. The conducted experiments show that the separation of the world model into an ego model and a model of the environment helps obtain policies that reach better performance in our experiments with highway driving. Decoupling these two models completely makes MPC perform faster and better, and helps to solve a stress-test scenario that requires quick reaction from the policy. We believe that our proposed approach can be applied to any problem that involves an agent with easily predictable kinematics acting in a complex stochastic environment, e.g. controlling robots in the real world, such as delivery carts or drones; or some Atari games, such as Space Invaders, or Freeway. We also believe that for the suggested separation to work, it is not strictly necessary to have the exact kinematic equations, the ego model can also be learned.

There is still more to explore about the proposed approaches. First, although DFM-KM MPC performs better than CFM-KM MPC, it is yet unclear if that is because of the modified cost function C^{km}, or because of the decoupled forward model. Experiments with a method that integrates C^{km} with CFM-KM PL would resolve this ambiguity. Second, comparing the results of CFM-KM PL and CFM-KM MPC, we see that policy learning performs much better, suggesting that DFM-KM PL is an approach worth investigating. Third, hand-designing the cost function is only possible for simple contexts, such as highway driving. For applications to more complex scenarios like urban driving, we would need to learn the cost function, which is highly non-trivial (Ng & Russell, 2000).

8 ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under NSF Award 1922658.

REFERENCES

Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L. Littman. Combating the compounding-error problem with a multi-step model. *arXiv:1905.13320 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1905.13320. arXiv: 1905.13320.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv:1710.11252 [cs]*, Mar 2018. URL http://arxiv.org/abs/1710.11252. arXiv: 1710.11252.

Luca Bergamini, Yawei Ye, Oliver Scheel, Long Chen, Chih Hu, Luca Del Pero, Blazej Osinski, Hugo Grimmett, and Peter Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. *arXiv:2105.12332 [cs]*, May 2021. URL http://arxiv.org/abs/2105.12332. arXiv: 2105.12332.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005.

Arthur E. Bryson, Yu-Chi Ho, and George M. Siouris. Applied optimal control: Optimization, estimation, and control. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(6):366–367, 1979. doi: 10.1109/TSMC.1979.4310229.

Gianluca Cesari, Georg Schildbach, Ashwin Carvalho, and Francesco Borrelli. Scenario model predictive control for lane change assistance and autonomous driving on highways. *IEEE Intelligent Transportation Systems Magazine*, 9(3):23–35, 2017. ISSN 1941-1197. doi: 10.1109/MITS.2017.2709782.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *arXiv:1802.07687 [cs, stat]*, Mar 2018. URL http://arxiv.org/abs/1802.07687. arXiv: 1802.07687.

Paul Drews, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg. Vision-based high speed driving with a deep dynamic observer. *arXiv:1812.02071 [cs]*, Dec 2018. URL http://arxiv.org/abs/1812.02071. arXiv: 1812.02071.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv:1812.00568 [cs]*, Dec 2018. URL http://arxiv.org/abs/1812.00568. arXiv: 1812.00568.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv:1406.2661 [cs, stat]*, Jun 2014. URL http://arxiv.org/abs/1406.2661. arXiv: 1406.2661.

David Ha and Jürgen Schmidhuber. World models. *arXiv:1803.10122 [cs, stat]*, Mar 2018. doi: 10.5281/zenodo.1207631. URL http://arxiv.org/abs/1803.10122. arXiv: 1803.10122.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv*, 2020. ISSN 23318422.

John Halkias and James Colyar. Ngsim interstate 80 freeway dataset, 2006. URL https://www.fhwa.dot.gov/publications/research/operations/06137/index.cfm. FHWA-HRT-06-137.

Tor Aksel N. Heirung, Joel A. Paulson, Jared O'Leary, and Ali Mesbah. Stochastic model predictive control — how does it work? *Computers & Chemical Engineering*, 114:158–170, 2018. ISSN 0098-1354. doi: https://doi.org/10.1016/j.compchemeng.2017.10.026.

Mikael Henaff, Junbo Zhao, and Yann LeCun. Prediction under uncertainty with error-encoding networks. *arXiv:1711.04994 [cs]*, Nov 2017. URL http://arxiv.org/abs/1711.04994. arXiv: 1711.04994.

Mikael Henaff, Alfredo Canziani, and Yann LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. *arXiv:1901.02705 [cs, stat]*, Jan 2019. URL http://arxiv.org/abs/1901.02705. arXiv: 1901.02705.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *arXiv:1903.00374 [cs, stat]*, Feb 2020. URL http://arxiv.org/abs/1903.00374. arXiv: 1903.00374.

Alexey Kamenev, Lirui Wang, Ollin Boer Bohan, Ishwar Kulkarni, Bilal Kartal, Artem Molchanov, Stan Birchfield, David Nistér, and Nikolai Smolyanskiy. Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation. *CoRR*, abs/2109.11094, 2021. URL https://arxiv.org/abs/2109.11094.

Jason Kong, Mark Pfeiffer, Georg Schildbach, and Francesco Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1094–1099, 2015. doi: 10.1109/IVS.2015.7225830.

S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pflueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, Sep 2008. ISSN 15564959, 15564967. doi: 10.1002/rob.20258.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv:1708.02596 [cs]*, Dec 2017. URL http://arxiv.org/abs/1708.02596. arXiv: 1708.02596.

Andrew Y. Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *in Proc. 17th International Conf. on Machine Learning*, pp. 663–670. Morgan Kaufmann, 2000.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. *arXiv:1507.08750 [cs]*, Dec 2015. URL http://arxiv.org/abs/1507.08750. arXiv: 1507.08750.

Brian Paden, Michal Cap, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *arXiv:1604.07446 [cs]*, Apr 2016. URL http://arxiv.org/abs/1604.07446. arXiv: 1604.07446.

Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control. *CoRR*, abs/2001.03093, 2020. URL http://arxiv.org/abs/2001.03093.

Oliver Scheel, Luca Bergamini, Maciej Wolczyk, Blazej Osinski, and Peter Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. *CoRR*, abs/2109.13333, 2021. URL https://arxiv.org/abs/2109.13333.

Georg Schildbach, Lorenzo Fagiano, Christoph Frei, and Manfred Morari. The scenario approach for stochastic model predictive control with bounds on closed-loop constraint violations. *Automatica*, 50(12):3009–3018, Dec 2014. ISSN 00051098. doi: 10.1016/j.automatica.2014.10.035. arXiv: 1307.5640.

Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. *arXiv:2101.06557 [cs]*, Jan 2021. URL http://arxiv.org/abs/2101.06557. arXiv: 2101.06557.

Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *arXiv:1711.03449 [cs, math]*, Jun 2018. URL http://arxiv.org/abs/1711.03449. arXiv: 1711.03449.

J. Ziegler, Moritz Werling, and Joachim Schroder. Navigating car-like robots in unstructured environments using an obstacle sensitive cost function. *2008 IEEE Intelligent Vehicles Symposium*, pp. 787–791, 2008.

A CALCULATING THE POLICY OUTPUT AGREEMENT ACROSS SEEDS

To measure the agreement across seeds in Table 2, we first take three policies that use the same method but different seeds and run them on 1000 examples from the dataset. We follow the procedure of Henaff et al. (2019) and, before unnormalizing models' output, clamp the values to the range $[-3, 3]$. For each method separately, we then calculate the mean and variance for the actions taken across seeds and data examples to obtain μ and σ^2. We then normalize the actions using these values. Now, for the normalized values, we calculate the standard deviation across outputs for different seeds for each dataset example separately. The values are then averaged across the entire 1000 examples to obtain the values reported in Table 2. Such a procedure accounts for the fact different methods output values of different magnitudes and avoids skewing the standard deviation comparison.

B IMPLEMENTATION DETAILS

Forward models To train updated forward models, we follow the procedure proposed by Henaff et al. (2019). The only change to the forward models is the change to the input dimension. The training method was unchanged.

CFM-KM Policy Training We use the same model as was proposed by Henaff et al. (2019), and train for 70 k steps, with batch size 10, and learning rate of 0.0001. We decrease the learning rate by a factor of 10 after 70% of training.

CFM-KM MPC To find the optimal action, we perform gradient descent for 11 iterations, with learning rate of 0.31. The cost function is calculated as: $C = C_{\text{proximity}} + 0.32 \cdot C_{\text{lane}} + 0.32 \cdot C_{\text{offroad}}$. The uncertainty cost proposed in Henaff et al. (2019) was not used for CFM-KM MPC as it made each iteration impractically slow. We used the plan length of 20 frames, which corresponds to 2 seconds. The hyperparameters were found with random search. We provide pseudo-code in figure 4b.

DFM-KM MPC To find the optimal action, we perform gradient descent for 27 iterations, with learning rate of 0.48. The cost function is calculated as: $C = 91.2 \cdot C_{\text{proximity}} + 2.88 \cdot C_{\text{offroad}} + 3.06 \cdot C_{\text{lane}} + 0.1 \cdot C_{\text{jerk}} + 0.001 \cdot C_{\text{destination}}$. We use the plan size of 30. The hyperparameters were found with random search. We provide pseudo-code in figure 4a.

We do not re-train the forward model for this setup, we simply use the CFM-KM, but we always run it with 0-actions — we get predictions $\boldsymbol{s}_{t+1}^{\text{env}}$ that correspond to what the world would have looked like if the ego-vehicle kept going at current speed and did not turn. We then assume that changing the actions would not have changed $\boldsymbol{s}_{t+1}^{\text{env}}$.

C OFFROAD COST

Another important difference between CFM PL and the other methods is that it does not use offroad cost C_{offroad}. The offroad cost component was introduced to prevent the ego-car from driving off the road. Without it, the cost of driving off the highway is the same as the cost of simply crossing a lane marker. To understand how much the offroad cost contributed to the improvement in performance, we test the CFM PL method with offroad cost. The results are presented in table 3. We see some improvement with adding offroad cost, particularly in interactive evaluation, but the performance does not reach the results of CFM-KM PL and DFM-KM MPC.

Input: Models f^{self} and f_θ^{env}
cost function C^{km}
states $\boldsymbol{s}_t^{\text{self}}$ and $\boldsymbol{s}_t^{\text{env}}$
planning horizon T
learning rate α
number of iterations N
Output: Action to be taken at time t
$\boldsymbol{a}_{t:t+T-1}^{\text{self}} \leftarrow 0$;
for $k \leftarrow 1$ **to** T **do**
  $\boldsymbol{s}_{t+k}^{\text{env}} \leftarrow f_\theta^{\text{env}}(\boldsymbol{s}_{t+k-1}^{\text{env}})$;
end
for $i \leftarrow 1$ **to** N **do**
  for $k \leftarrow 1$ **to** T **do**
    $\boldsymbol{s}_{t+k}^{\text{self}} \leftarrow f^{\text{self}}(\boldsymbol{s}_{t+k-1}^{\text{self}}, \boldsymbol{a}_{t+k-1}^{\text{self}})$;
  end
  $J \leftarrow \sum_{k=1}^{T} \gamma^k C^{\text{km}}(\boldsymbol{s}_{t+k}^{\text{env}}, \boldsymbol{s}_{t+k}^{\text{self}}, \boldsymbol{a}_{t+k-1}^{\text{self}})$;
  $\boldsymbol{a}_{t:t+T-1}^{\text{self}} \leftarrow \boldsymbol{a}^{\text{self}} - \alpha \frac{\partial J}{\partial \boldsymbol{a}_{t:t+T-1}^{\text{self}}}$;
end
return $\boldsymbol{a}_t^{\text{self}}$;

(a) DFM-KM MPC

Input: Models f^{self} and f_θ^{env}
cost function C
states $\boldsymbol{s}_t^{\text{self}}$ and $\boldsymbol{s}_t^{\text{env}}$
planning horizon T
learning rate α
number of iterations N
Output: Action to be taken at time t
$\boldsymbol{a}_{t:t+T-1}^{\text{self}} \leftarrow 0$;
for $i \leftarrow 1$ **to** N **do**
  for $k \leftarrow 1$ **to** T **do**
    $\boldsymbol{s}_{t+k}^{\text{self}} \leftarrow f^{\text{self}}(\boldsymbol{s}_{t+k-1}^{\text{self}}, \boldsymbol{a}_{t+k-1}^{\text{self}})$;
    $\boldsymbol{s}_{t+k}^{\text{env}} \leftarrow f_\theta^{\text{env}}(\boldsymbol{s}_{t+k-1}^{\text{env}}, \boldsymbol{s}_{t+k-1}^{\text{self}})$;
  end
  $J \leftarrow \sum_{k=1}^{T} \gamma^k C^{\text{km}}(\boldsymbol{s}_{t+k}^{\text{env}}, \boldsymbol{s}_{t+k}^{\text{self}}, \boldsymbol{a}_{t+k-1}^{\text{self}})$;
  $\boldsymbol{a}_{t:t+T-1}^{\text{self}} \leftarrow \boldsymbol{a}^{\text{self}} - \alpha \frac{\partial J}{\partial \boldsymbol{a}_{t:t+T-1}^{\text{self}}}$;
end
return $\boldsymbol{a}_t^{\text{self}}$;

(b) CFM-KM MPC

Figure 4: Algorithms of the proposed MPC methods. Note that DFM-KM-MPC runs f_θ^{env} outside the main optimization loop, while CFM-KM-MPC runs it inside, causing it to take more time per iteration.

Table 3: Comparison of CFM PL with and without offroad cost component.

		Interaction Policy	
Method	**Fixed Replay**	**CFM-KM Policy**	**CFM Policy**
CFM Policy	25.2 ± 3.0	7.3 ± 2.1	4.8 ± 2.3
CFM Policy with C_{offroad}	25.4 ± 1.6	2.1 ± 0.8	3.3 ± 0.2

D DFM-KM MPC COST

The cost calculation consists of two stages, as described in Section 4: mask creation and cost calculation.

Mask creation We create two kinds of masks: one for proximity cost, and the other for offroad and lane costs. The masks are created in a way to align with the predicted center of the car and face the direction of the car's heading. To create masks given the position relative to the center of the image, direction, speed, width, and length of the agent $(x, y, u_x, u_y, s, w, l)$ we follow the steps below:

1. Create a mesh grid of coordinates. The rasterized image resolution we use is 117 by 24, and the size of the corresponding area is 72.2 by 14.8 meters. We first create a matrix of coordinates of each of the cells in the image with respect to the center of the image. $A : \mathbb{R}^{117\times24\times2}$, $A_{i,j} = [(72.2/117 \cdot i - 36.1)), (14.8/24 \cdot j - 12.4)]$.
2. In order to align the coordinates with the ego-car, we shift and rotate them: $B_{i,j} = R^{u_x,u_y}[A_{i,j,1} - x, A_{i,j,2} - y]$, where R^{u_x,u_y} is the rotation matrix for the angle specified by (u_x, u_y).
3. Construct the masks. First, we define the safety distance along the direction of movement: $d_x = 1.5 \cdot (\max(10, s) + l) + 1$, and in the orthogonal direction: $d_y = w/2 + 3.7$. 3.7

Table 4: Comparison of characteristics of the tested methods.

Method	Decoupled Forward Model	Kinematics Model	Learned Policy	Modified cost
CFM Policy	✗	✗	✓	✗
CFM-KM Policy	✗	✓	✓	✗
CFM-KM MPC	✗	✓	✗	✗
DFM-KM MPC	✓	✓	✗	✓

here is the lane width. These are the distances beyond which the objects are not taken into account in the cost. Then, we want to build a mask that reaches 0 at d_x in the front and behind, at d_y at the sides, and reaches 1 at the car edges. The masks are:

$$M^{\text{car}}_{i,j} = \left[\left(\frac{d_x - |B_{i,j,1}|}{d_x - l/2} \right)^+ \cdot \min \left(\left(\frac{d_y - |B_{i,j,2}|}{d_y - w/2} \right)^+, 1 \right) \right]^\alpha \tag{5}$$

$$M^{\text{side}}_{i,j} = \left[\left(\frac{d_x - |B_{i,j,1}|}{d_x - l/2} \right)^+ \cdot \left(\frac{d_y - |B_{i,j,2}|}{d_y - w/2} \right)^+ \right]^\alpha \tag{6}$$

The difference between M^{side} and M^{car} are in the clamping values above 1: when calculating the cost component accounting for proximity to other vehicles, we would like the mask profile to have a "flat nose". α is a hyperparameter used to make the mask non-linear. Higher values make the cost grow more rapidly as objects come closer to the ego-vehicle.

Note that all operations are differentiable with respect to (x, y, u_x, u_y, s), allowing us to backpropagate through $\boldsymbol{s}^{\text{self}}$.

Calculating cost components Having the masks, we simply perform element-wise multiplication with the corresponding channels of the $\boldsymbol{s}^{\text{env}}$, see figure 5.

$$C_{\text{lane}} = \langle \boldsymbol{s}^{\text{env}}_{\text{lanes}}, M^{\text{side}} \rangle \tag{7}$$

$$C_{\text{offroad}} = \langle \boldsymbol{s}^{\text{env}}_{\text{offroad}}, M^{\text{side}} \rangle \tag{8}$$

$$C_{\text{proximity}} = \langle \boldsymbol{s}^{\text{env}}_{\text{car}}, M^{\text{car}} \rangle \tag{9}$$

$$\tag{10}$$

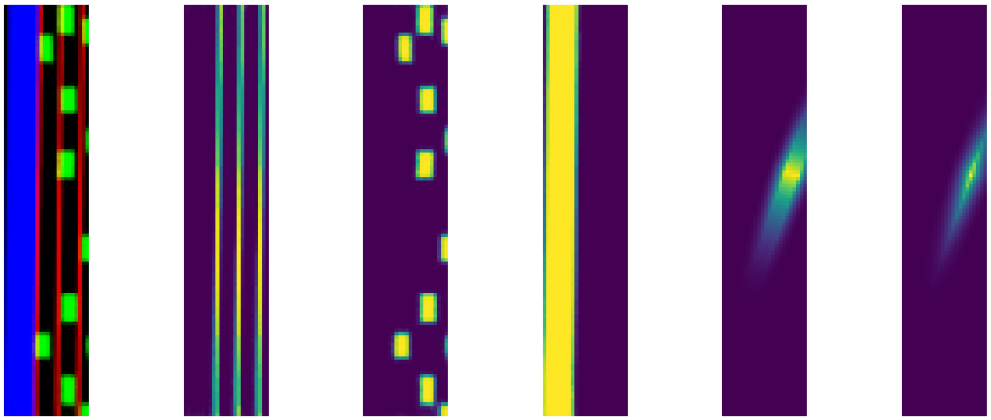
The new cost also introduces two new cost components: destination cost, and jerk cost. Destination cost is simply a term that pushes the car to go forward. This prevents cases where MPC fails to drive forward because there are no cars behind. The cost is calculated as $C_{\text{destination}} = -x$. Jerk cost is responsible for making the actions more smooth. Intuitively, this is a cost that penalizes the derivative of the actions w.r.t. time. $C_{\text{jerk}} = \frac{1}{T} \sum_{t=2}^{T} (\boldsymbol{a}^{\text{self}}_t - \boldsymbol{a}^{\text{self}}_{t-1})^\top (\boldsymbol{a}^{\text{self}}_t - \boldsymbol{a}^{\text{self}}_{t-1})$.

The components are then combined into a single scalar with the corresponding weights α:

$$\begin{aligned} C = & \alpha_{\text{lane}} C_{\text{lane}} + \\ & \alpha_{\text{offroad}} C_{\text{offroad}} + \\ & \alpha_{\text{proximity}} C_{\text{proximity}} + \\ & \alpha_{\text{destination}} C_{\text{destination}} + \\ & \alpha_{\text{jerk}} C_{\text{jerk}} \end{aligned} \tag{11}$$

E COMPARISON OF METHODS

In table 4 we show the proposed methods and the comparison of the used components.



(a) $\boldsymbol{s}^{\text{env}}$ (b) $\boldsymbol{s}^{\text{env}}_{\text{lanes}}$ (c) $\boldsymbol{s}^{\text{env}}_{\text{cars}}$ (d) $\boldsymbol{s}^{\text{env}}_{\text{offroad}}$ (e) M^{car} (f) M^{side}

Figure 5: **Components used for calulating costs.** Figure 5a depicts the original image, and 5b, 5c, 5d depict channels used for cost calculation. 5e shows the mask used for car proximity cost, while 5f is used for offroad and lane costs respectively.