Physically Feasible Vehicle Trajectory Prediction

Harshayu Girase[†§*] **Jerrick Hoang**[†*] **Sai Yalamanchi**[†] **Micol Marchetti-Bowick**[†]

[†] Uber ATG [§] UC Berkeley

Abstract

Predicting the future motion of actors in a traffic scene is a crucial part of any autonomous driving system. Recent research in this area has focused on trajectory prediction approaches that optimize standard trajectory error metrics. In this work, we describe three important properties – physical realism guarantees, system maintainability, and sample efficiency – which we believe are equally important for developing a self-driving system that can operate safely and practically in the real world. Furthermore, we introduce **PTNet** (PathTrackingNet), a novel approach for vehicle trajectory prediction that is a hybrid of the classical pure pursuit path tracking algorithm and modern graph-based neural networks. By combining a structured robotics technique with a flexible learning approach, we are able to produce a system that not only achieves the same level of performance as other state-of-the-art methods on traditional trajectory error metrics, but also provides strong guarantees about the physical realism of the predicted trajectories while requiring half the amount of data. We believe focusing on this new class of hybrid approaches is an useful direction for developing and maintaining a safety-critical autonomous driving system.

1 Introduction

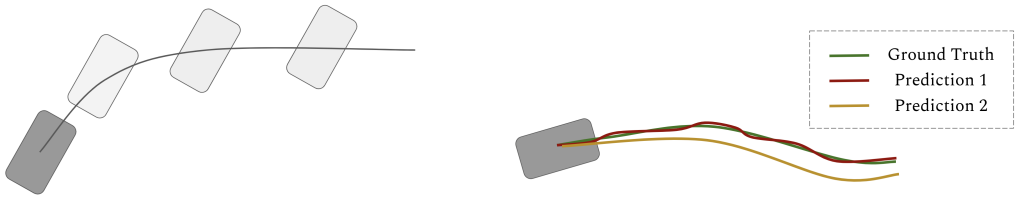
Over the past several years, significant advancements have been made in autonomous driving. One active area of research is in methods for predicting the future motion of surrounding actors in the scene. In this work, we focus on the prediction problem, and in particular, we want to direct the attention of the research community to a less explored approach: combining robotics with machine learning. We argue that this hybrid approach is useful in order to maintain a safety critical system over a long period of time. In particular, we focus on the following three aspects.

Physical realism guarantees: Despite being a very powerful toolkit, pure machine learning methods provide few theoretical guarantees with regard to the physical dynamics of the predicted trajectories. On the other hand, robotic techniques for understanding rigid body movement have been extensively studied and theoretically understood [1, 2]. In this paper, we propose a hybrid technique, leveraging the power of learning from data along with the advantage of strong theoretical guarantees that robotic techniques provide. We also emphasize the importance of evaluating prediction quality based on trajectory feasibility metrics in addition to trajectory error metrics. Figure 1 shows two hypothetical situations where trajectory error metrics do not paint a full picture. We show quantitatively and qualitatively that our model outperforms state-of-the-art models on key feasibility metrics.

System maintainability: Maintainability is the degree to which an application is understood, or the ease with which an application can be maintained, enhanced, and improved over time. Using the framework of *technical debt*, the authors of [3] argue that despite the quick wins machine learning methods bring to a system, it is remarkably easy to incur a massive maintenance cost over the long

∗ indicates equal contribution

Machine Learning for Autonomous Driving Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.



(a) A hypothetical scenario where the predicted trajectory has zero displacement error. However, the predicted orientation makes the trajectory dynamically infeasible for a vehicle to follow.

(b) A hypothetical scenario where prediction 1 (red) has smaller trajectory error than prediction 2 (yellow) in comparison to ground truth (green). However, the red predicted trajectory is physically unrealistic.

Figure 1: Two scenarios scenarios demonstrating that trajectory error metrics do not fully capture the important characteristics of the predicted trajectories.

run. Specifically, the authors emphasize boundary erosion, arguing that machine learning models create entanglement which make isolated improvements impossible. Combining robotic techniques with machine learning naturally introduces stronger abstraction boundaries, creating an encapsulated and modular design that leads to more maintainable code in the long run [4].

Sample efficiency: It is expensive to collect a large amount of real-world driving data, particularly if the autonomous driving system also depends on having a high-definition map. Machine learning methods require a large amount of data in order to learn high-dimensional correlations from scratch. Robotic techniques, on the other hand, already incorporate distilled knowledge about the world in the form of physics equations. A hybrid approach increases sample efficiency while still learning from data. We show that our model is twice as sample efficient and still outperforms or matches state-of-the-art performance in trajectory error by explicitly modeling and executing motion profiles.

Overall, in order to develop and maintain a safety critical autonomous driving system, we believe that it is useful to explore more structured machine learning approaches by combining them with existing and well-studied robotics techniques. In this work, we combine graph neural networks with the Pure Pursuit path tracking algorithm [5], leveraging the power of data to learn a motion profile predictor for each actor and executing the motion profile with a Pure Pursuit model. We show quantitatively and qualitatively that our method achieves the same level of performance as state-of-the-art methods, while also providing strong physical realism guarantees and improving on sample efficiency.

2 Related Work

With the recent success of deep learning in many domains, recent work has focused on deep neural network based approaches for trajectory prediction. Specifically, the recurrent neural network (RNN) has shown promising performance for sequence learning and many works have employed long short-term memory (LSTM) and gated recurrent unit (GRU) networks for future motion forecasting given past sequential observations [6, 7, 8, 9, 10]. In order to capture surrounding information, recent papers have proposed to use contextual scene information for scene-compliant prediction [11, 12, 13, 14]. Most of these methods encode scene context features using convolutional neural networks (CNN) and then use learnt embeddings downstream to predict actor trajectories. Another family of deep-learning-based approaches has explored graph neural networks to model interactions between agents [15, 16, 17, 18, 19, 20]. These deep networks learn vehicle dynamics, such as feasible steering and acceleration, from the large number of example trajectories available during training. However, unconstrained motion prediction may violate the dynamic feasibility constraints of vehicles. Most of these methods only predict the center positions of a vehicle or independently predict [21, 22] position and heading which may still result in infeasible discrepancies between the two, and as a result may poorly capture the vehicle's occupancy in space.

Other approaches also introduce structure into the task by exploring goal-conditioned ideas which model actor intent and goals prior to predicting the trajectory. Rhinehart et al. [23] propose PRECOG, a goal-conditioned approach for multi-agent prediction. More recently, Mangalam et al. [24] propose an endpoint conditioned prediction scheme which conditions pedestrian predictions on goal destinations. In the vehicle setting, GoalNet [25] uses lane centerlines as goal paths and predicts trajectories in the path-relative Frenet frame. The authors of MultiPath [26] similarly introduce

the idea of trajectory anchors to provide more structure for the problem. While in practice these structured approaches could provide more system maintainability and sample efficiency, they still do not provide an absolute guarantee on the dynamic feasibility of the predicted trajectories since the predicted offsets could be arbitrary. Our method leverages the goal-based trajectory prediction approach introduced in [25], which allows long-term behavior to be captured. However, we also incorporate a path tracking algorithm that provides strong guarantee on the physical realism of the resulting trajectories.

There have been a few prior approaches that combine robotics techniques with machine learning. These methods have focused primarily on state estimation or short-term instantaneous trajectory prediction [27, 28] through kinematic-based vehicle models such as a Kalman Filter. While these models work well for short-term prediction, they become increasingly unreliable for longer-term prediction as the vehicle will change control inputs based on scene elements or other vehicles. The majority of robotics-focused research that has been applied to the autonomous driving domain has been on the motion planning side, e.g. [29, 30, 31]. In contrast, our approach combines learning and robotics for the prediction task. CoverNet [32] is a recent approach that ensures physical realism by turning the trajectory prediction problem into a classification task over a set of dynamically feasible trajectories. However, in order to achieve good coverage over the space of possible future motion, a large number of trajectories must be generated, which might be impractical for a system that must run in real time. The deep kinematic model (DKM) introduced by Cui et al. [22] is closest to our philosophy, as they predict control inputs such as acceleration and steering prior to trajectory roll-out. While they ensure dynamic feasibility of their trajectories, their proposed method does not explicitly use goal path information to guide predictions that may be useful for longer-horizon predictions. Our work relies on Pure Pursuit path tracking [5] to produce goal-directed trajectories that yield better long-term predictions while still providing strong guarantees on curvature and higher-order dynamics.

3 Method

In this section, we present our method for combining a flexible data-driven model with a structured path tracking algorithm to produce dynamically feasible trajectory predictions for vehicles. Our approach is built on top of GoalNet, a goal-based trajectory prediction model introduced in [25]. The key difference between our model and GoalNet is that we replace the final trajectory prediction layer with a highly structured Pure Pursuit path tracking (PT) layer. We therefore call our model PTNet.

In this work, we focus solely on the trajectory prediction problem. We assume that there is a real-time detection and tracking system onboard the SDV to detect and estimate the states of surrounding traffic actors. For each actor, our model is given the actor's dynamics state (current heading, position, velocity, and acceleration) and its past positions. Furthermore, mapped lane boundaries and traffic sign locations are also available as inputs. Given this information, our model then makes multi-modal spatio-temporal predictions of the actor's positions over the next T timesteps.

3.1 Background

Our method is built on the path generation and graph encoder layers introduced in GoalNet [25] combined with the Pure Pursuit algorithm [5]. In the remainder of this section, we provide some background on each of these building blocks.

3.1.1 Path Generation

The path generation module generates a set of map-based goal paths and one additional map-free path for each actor. To generate map-based paths, we treat the map as a lane graph, where each node is a lane (with no branching points) and a directed edge connecting two nodes if the source node is the predecessor lane of the target node. Given the lane graph and the actor's current position, we query for the closest nodes (we use a radius $r = 2$m) and roll out the paths by traversing the graph and connecting the centerlines of the lanes. The result is a list of lane sequences we call *map-based paths*. Since the map might not capture the high-level intention of the actor, we also generate one additional *map-free path* in the direction of travel of the actor. For each actor A^i, we then transform each goal path G^i_j into the actor's frame of reference. These will later be used as reference paths for our path tracking algorithm.

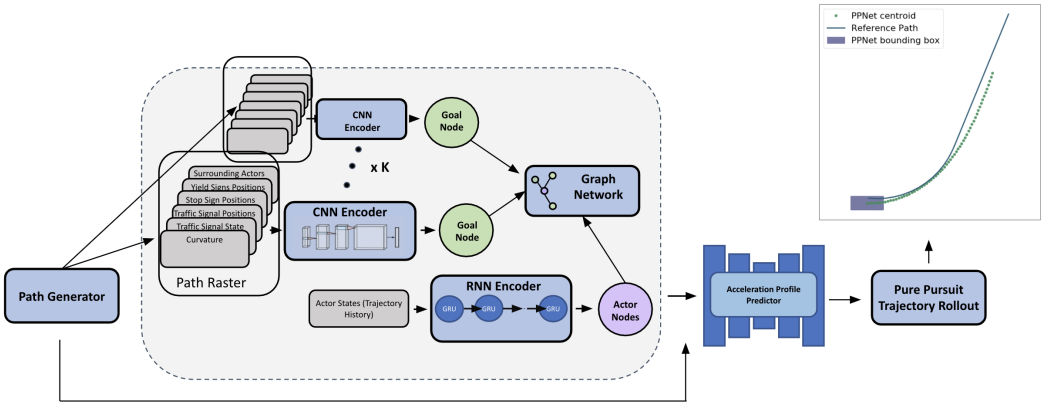


Figure 2: An overview of our method. Given a high-definition map and a perception system that outputs tracked objects, our method consists of three main steps. (1) Path generation: for each object, we generate a set of goal paths. (2) Input encoding: for each actor, we construct a graph that consists of one single actor node and a variable set of goal nodes, and then encode the actor state and the goal information into latent features using a GNN. (3) Trajectory development: for each goal, we predict N temporal motion profiles and execute each motion profile using the PurePursuit path tracking algorithm to produce a trajectory.

3.1.2 Graph Networks Encoder

To handle a variable number of goals for each actor, we use a graph network [33] following the method introduced in GoalNet [25]. In particular, we construct a mini graph for each actor. The graph consists of a variable number of goal nodes and a single actor node. The actor's current and past states are encoded into the initial actor node features. The path and context information along each path, e.g. signage locations and curvature, is encoded into initial goal node features. For the initial set of features for each edge corresponding to a goal, we use the actor's current velocity and acceleration to construct a 0-jerk rollout and project it to onto the path. We then follow the equation proposed in [25] for our graph network updates.

Specifically, for each actor i, denote v_i^ℓ to be the encoded actor node attributes at layer ℓ. Let the set of all goals for each i be G_i. For each goal $j \in G_i$ of actor i, denote e_{ij}^ℓ to be the edge attributes at layer ℓ and let g_j be the goal node attributes for the goal j. Note that g_j does not have a layer index since the goal nodes do not have incoming edges so they are not updated at each iteration. The update equations are given by,

$$e_{ij}^{\ell+1} = \phi_e(v_i^\ell, e_{ij}^\ell, g_j) \tag{1}$$

$$v_i^{\ell+1} = \phi_v(v_i^\ell, \psi(\{e_{ij}^\ell\}_{j \in G_i})) \tag{2}$$

We use a 2-layer MLP for ϕ_v and ϕ_a and the mean function for ψ. This simple graph network contains a total of 2 layers.

3.1.3 Pure Pursuit Algorithm

Pure Pursuit [5] is a simple path tracking algorithm that calculates the necessary arc to reach a "goal point" on the path. The algorithm is aptly named based on the way humans drive – we look at a point we want to go to and control the car to reach that point. The high level steps of the Pure Pursuit algorithm are outlined in Figure 7 and are described in detail below.

Finding the target point on the path: Given the actor's position and a goal path, the algorithm first tries to find a target point on the path to track. In particular, given a fixed lookahead distance L, we find all possible intersections of the circle of radius L centering at the actor's control point with the goal path. Our paths are generated so that they are sufficiently close to the actor such that an initial intersection point always exists. In the case where there is more than one intersection, we only consider the point ahead of the actor on the path. In all of our experiments, we use $L = 10$m.

Assume circular motion and calculate curvature: Assuming circular motion, we find the circle passing through the goal point and tangent to the control point in the direction of heading. Given

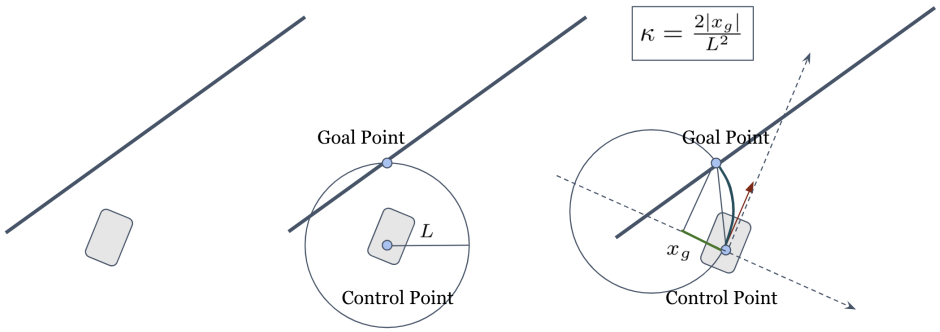


Figure 3: An overview of the Pure Pursuit path tracking update.

the geometric setup, it can be proved (see [5]) that the curvature can be obtained by $\kappa = {}^{2|x_g|}/_{L^2}$ where x_g is the x-component of the goal point in the actor's frame. However, this curvature can be infeasible. To ensure that we only execute feasible motion, we calculate the final curvature by $\kappa = \min(2|x_g|/L^2, M_c)$ where M_c is the maximum possible curvature that can be executed by a vehicle. In our experiments, we choose $M_c = 0.3$.

Update path tracking state: Assume the state is comprised of position $p_t^i = (x_t^i, y_t^i) \in \mathbb{R}^2$, velocity $v_t^i \in \mathbb{R}^2$, and heading $h_t^i \in [-\pi, \pi)$. We use the following state update equations. The input acceleration at each time step is predicted by the acceleration profile predictor described in the previous section.

$$\begin{cases} x_{t+1}^i = x_t^i + \cos(h_t^i) v_t^i \Delta_t \\ y_{t+1}^i = y_t^i + \sin(h_t^i) v_t^i \Delta_t \\ v_{t+1}^i = v_t^i + a_t^i \Delta_t \\ \kappa = \min(2x_g/L^2; M_\kappa) \\ h_{t+1}^i = h_t^i + v_t^i \Delta_t \kappa \end{cases}$$

3.2 PTNet

Our main contribution is connecting the idea of map-adaptive reference path generation and scoring introduced by [25] with a structured path following algorithm introduced by [5]. The outline of our method is described in Figure 2. The first two components, path generation and graph network encoding, are taken from GoalNet. The last component is a differentiable Pure Pursuit layer. We connect these two components using a temporal profile predictor. This layer also serves as a semantically meaningful abstraction between the two components. Specifically, the output of this layer is a sequence of acceleration values over time which can be visualized and has a specific physical meaning, unlike most intermediate layers in a fully learned system. We implemented all operations such that the system remains end-to-end trainable. The predicted acceleration profile as well as curvature update in PurePursuit provides theoretical guarantees on physical realism. Also, by directly leveraging a motion model, we provide a starting point from which the model can learn, making it more sample efficient.

3.2.1 Multi-Modal Acceleration Profile Generation

For each of our spatial modes (anchored by reference paths), we generate N different temporal modes. Temporal modes can simply be modeled by generating multiple different acceleration profiles to be inputted into our path tracking algorithm. We follow the same unsupervised training scheme proposed in [11] to learn different acceleration profile modes. In particular, we have N acceleration prediction networks, each learning a different motion profile, and a mode prediction network to learn the probabilities for predicting each mode. On each iteration, only the mode whose output is closest to the ground truth is penalized in the loss. To ensure physical realism, we constrain the output of each network to be within -8 m/s^2 and 8 m/s^2 by using a scaled $\tanh$ activation. The acceleration prediction layer serves as a strong abstraction between input encoding and path tracking layer, which

enforces system maintainability. By predicting in control space as an intermediate step instead of predicting position directly, the model also provides a semantically meaningful intermediate layer which improves interpretability.

3.2.2 Loss Function

The loss function used to train this model is a combination of the mode classification loss and trajectory regression loss. For each actor i, we identify the set of goals ${\mathcal{G}^*}_i$ that match the future ground truth trajectory of the actor (we use the same path labeling algorithm defined in [25]). For each actor i, we define the target probability of $1/|{\mathcal{G}^*}_i|$ for each goal. For each matching goal, we assign target probability of 1 for the best matching temporal mode. The target probability of a trajectory mode is the product of the target probability of the spatial (goal) mode and the target probability of the temporal mode (one-hot). For each actor we also calculate the $L1$ smooth loss between the ground-truth trajectory and the trajectory modes weighted by probability,

$$\mathcal{L} = \sum_{i \in \mathcal{A}} [\underbrace{(- \sum_{m \in \mathcal{T}^i} p_m \log \hat{p}_m)}_{\text{Mode Classification Loss}} + \underbrace{(\sum_{m \in \mathcal{T}^i} p_m ||\tau_i - \hat{\tau}_m||_1)}_{\text{Trajectory Error Loss}}]$$

Here we denote the set of all actors to be $\mathcal{A}$ and the set of all spatio-temporal trajectory modes for an actor i to be $\mathcal{T}^i$. We define the target probability mass for mode m to be p_m and the predicted mass to be $\hat{p}_m$. We denote the predicted trajectory for temporal mode m to be $\hat{\tau}_m$ and the ground truth trajectory of actor i to be τ_i.

4 Experiments

4.1 Datasets

We evaluate our method on two datasets: our internal dataset and the public NuScenes dataset [34]. The NuScenes dataset has 1.4M objects over 40K frames collected from 15 hours of driving in Boston and Singapore. Our internal dataset has 138M objects (60% of which are vehicles) over 6M frames and was collected from various cities in the United States. We only consider non-parked vehicles and those for which we observe at least 6 seconds of future.

4.2 Baselines

We compare our model with 4 different baselines: GoalNet [25], Multiple Trajectory Prediction (MTP) [11], MultiPath [26] and CoverNet [32]. For CoverNet, we use the static version of CoverNet which relies on having a predefined set of trajectories. We directly use the publicly released set of 2206 trajectories for the NuScenes dataset. We did not compare to CoverNet on our internal dataset. MTP predicts a fixed number of different modes and encourages diversity by only updating the winning mode. In our experiments, we use 3 modes for MTP. MultiPath uses a fixed-size set of spatial-temporary trajectory anchors which are estimated by running a clustering algorithm on the training dataset. The method then makes trajectory predictions by outputting offsets from the best anchor. We use 64 modes for our MultiPath comparison.

4.3 Feasibility Metrics

In addition to evaluating our method and baselines on standard trajectory error metrics, we also evaluate them on other metrics that capture physical realism. To define the metrics, we first separate the concept of heading into *bounding box heading* and *motion heading*. An actor's bounding box heading is the direction the actor is facing. In contrast, motion heading is defined to be the direction between two consecutive control points. For our model, these concepts are the same. However, we want to define a set of metrics that can be broadly applicable. Next, we define four key directions that will be used to compute the feasibility metrics. The longitudinal direction is defined to be the direction of the bounding box heading. The lateral direction is defined to be the direction orthogonal to the longitudinal direction. The traversal direction is defined to be the direction of the motion heading. The centripetal direction is defined to be the direction orthogonal to the traversal direction. Using these concepts, we define a set of physical feasibility metrics below.

Method	avg ATE	avg CTE	avg DE
PTNet-1T	**1.82**	0.51	**2.04**
GoalNet-1T	1.84	**0.48**	2.05
MTP	2.44	0.77	2.77
MultiPath	3.62	0.90	4.01
CoverNet	4.07	1.14	4.57

(a) Comparison of the most probable trajectory error on the NuScenes dataset.

Method	avg ATE	avg CTE	avg DE
PTNet-1T	**2.15**	0.49	2.38
GoalNet-1T	2.16	**0.47**	**2.37**
MTP	2.39	0.59	2.67
MultiPath	2.59	0.66	2.91
CoverNet	—	—	—

(b) Comparison of the most probable trajectory error on our internal dataset. CoverNet results are not available since CoverNet requires a trajectory anchor set that is specific to the dataset.

Table 1: Trajectory error metrics comparison on the public NuScenes dataset and our internal dataset.

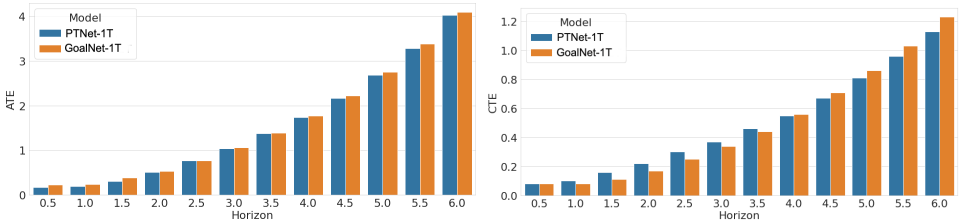


Figure 4: Trajectory error vs horizon comparison between PTNet-1T and GoalNet-1T. PTNet outperforms GoalNet-1T in along-track error due to its ability to explicitly model the acceleration profile for each goal.

Curvature Violation: The curvature over a trajectory segment is given by $\kappa = {}^{2\sin\Delta h^i}/_{\Delta p^i}$, where Δh^i denotes the change in bounding box heading and Δp^i denotes the change in position between two waypoints. A trajectory contains a curvature violation if $\kappa > 0.3$ m^{-1} at any point along the trajectory.

Lateral Speed Violation: If we decompose the speed vector v_t^i into its lateral and longitudinal components, a trajectory contains a lateral speed violation if at any given point t, the instantaneous lateral speed is greater than 1 m/s.

Centripetal Acceleration Violation: If we decompose the acceleration vector a_t^i into its traversal and centripetal components, a trajectory contains a centripetal acceleration violation if at any given point t, the instantaneous centripetal acceleration is greater than 10 m/s^2.

Traversal Acceleration Violation: If we decompose the acceleration vector a_t^i in to its traversal and centripetal components, a trajectory contains a traversal acceleration violation if at any given point t, the instantaneous centripetal acceleration is smaller than -12 m/s^2 or greater than 8 m/s^2.

All of the above boundary constraints are chosen from studying the physical constraints (acceleration, turning radius) of a standard mid-size SUV. We also made sure that none of the ground truth trajectories in our datasets violate any of these constraints.

4.4 Results

We first compare our method with all baselines on a set of general trajectory error metrics. For PTNet and GoalNet, we use the suffix "-NT" to refer to a model with N temporal modes. In particular, we report the cross-track (CTE), along-track (ATE) and displacement error (DE) on the most probable trajectory in both our internal dataset and NuScenes dataset. The results are shown in Table 1. In this table, the errors are averaged over all horizons. The precise definitions of these metrics can be found in the Appendix. We observe that PTNet performs on par with GoalNet on these trajectory error metrics while beating all other baselines across all metrics.

Next, we dive deeper into comparing PTNet-1T and GoalNet-1T. Specifically, we report the along-track and cross-track errors of the predicted trajectory mode that best matches the ground truth against the prediction horizon. The results are shown in Figure 4. We observe that on best matching error metrics, PTNet-1T outperforms GoalNet-1T on along-track error across all horizons. This could be

Method	Curvature Violations	Lateral Speed Violations (%)	Centripetal Accel Violations (%)	Min Traversal Accel Violations (%)	Max Traversal Accel Violations (%)
GoalNet-1T	65.16	0.43	16.64	0.06	0.76
GoalNet-2T	68.80	4.41	22.34	1.90	3.23
GoalNet-3T	69.02	6.71	25.86	3.60	5.64
PTNet-1T	0	0.27	9.49	0	0
PTNet-2T	0	0.27	9.21	0	0
PTNet-3T	0	0.27	7.90	0	0
GroundTruth	0	0	0	0	0

Table 2: Trajectory feasibility metrics comparison on the public NuScenes dataset.

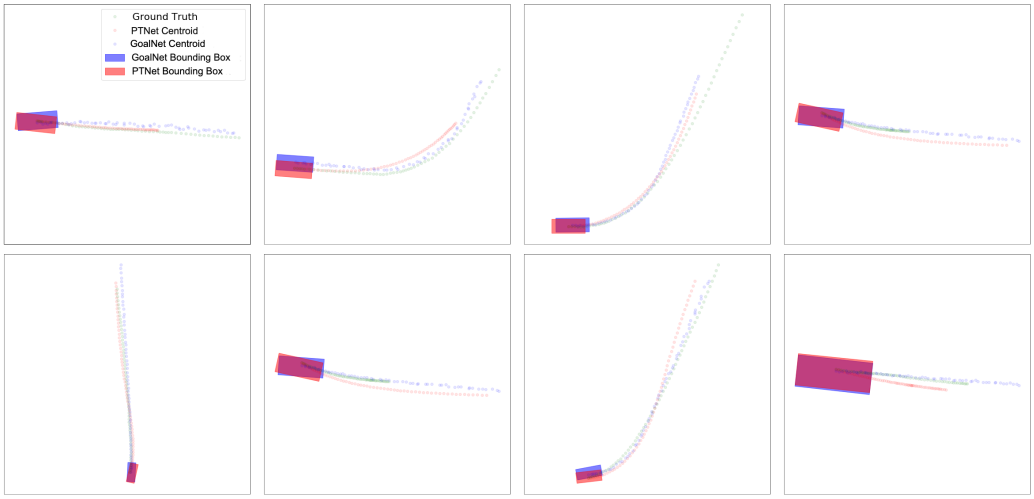


Figure 5: Qualitative results comparing GoalNet-1T and PTNet-1T. We can see that PTNet's output is always smooth, while GoalNet's can be noisy and physically infeasible.

attributed to PTNet's explicit representation of the acceleration profile, which makes the task easier to learn. On cross-track error, PTNet outperforms GoalNet in later horizons (> 3.5 seconds) but is worse in shorter horizons (< 3.5 seconds). In general, the short-term behavior of an actor is much easier to model with fewer constraints, while long-term behavior is easier to model with more structure. This is why we posit incorporating the structure from a robotics-based approach helps PTNet outperform GoalNet in the longer horizons.

Next, we compare our method to GoalNet on the feasibility metrics described in Section 4.3. We omit a comparison with other baselines on these metrics because GoalNet is the strongest baseline in terms of trajectory error and it provides the most direct comparison for evaluating the impact of our structured "robotics" layer on the dynamic feasibility of the resulting trajectories. We report feasibility results on the NuScenes dataset. In this experiment, we trained GoalNet and PTNet using three different choices of temporal modes, denoted by 1T,2T,3T. For each model and for each metric, we report the fraction of total trajectories violating the metric. We count one trajectory as one violation regardless of the number of waypoints violating the requirement. The result is shown in Table 2. We observe that all variants of PTNet perform significantly better than all variants of GoalNet across all metrics. Specifically, because of the way PTNet is set up, curvature and traversal acceleration requirements are guaranteed to be satisfied. We confirm quantitatively that PTNet shows 0% violations in these categories. We also observe that as we increase the number of temporal modes, i.e. increasing the number of trajectories produced per goal, GoalNet tends to produce more trajectories violating physical realism while the PTNet numbers stay relatively constant or decrease slightly. We also show qualitative plots in Figure 5. Qualitatively, we observe that PTNet's trajectory outputs, although sometimes further away from the ground truth in terms of L2 distance, are always smooth. In contrast, GoalNet is slightly more accurate in terms of average trajectory error metrics, but its trajectories can be noisy and non-smooth.

Lastly, we experiment on sample efficiency with GoalNet-1T and PTNet-1T. We set up the experiment by evaluating each model when trained on X% of the training set and evaluating on the entire test

Figure 6: A comparison of the sample efficiency of PTNet-1T and GoalNet-1T. We plot the best-matching average displacement error against percentage of training data. We observe that PTNet can be up to 2 times more sample efficient to reach the same level of performance.

set. For each value of X, we use four different random seeds to randomly select X% of the training logs. We train both models on this exact set of sampled logs and evaluate on the test set. In this experiment, we report the performance of GoalNet-1T and PTNet-1T using the best matching average displacement error. Results are shown in Figure 6. We can see that because PTNet can leverage the distilled knowledge about the world through the dynamics equation update, the model requires 2X less data in order to converge to the same level of performance as GoalNet. On the other hand, GoalNet needs to learn physics from scratch, which requires a lot more data.

Overall, we observe that the addition of a robotics layer helps PTNet produce smooth and dynamically feasible trajectories while maintaining state of the art level performance on trajectory error metrics and requiring 2X less data.

5 Conclusion

In this work, we introduce PTNet which combines a machine-learning-based trajectory prediction model with a structured path-tracking algorithm. Specifically, we introduce a differentiable path-tracking (PT) layer that can be added to existing architectures while still allowing end-to-end training. PTNet models and executes one or more acceleration profiles per path using a differentiable Pure Pursuit path tracker. The final output is a set of multi-modal spatial-temporal trajectories. By leveraging the power of learning from data as well as the structure embedded in classical physics-based robotics methods, PTNet is able to significantly improve on physical realism of the predicted trajectories while maintaining state-of-the-art performance on key trajectory error metrics and requiring two times less data. We view this as a small step towards a bigger idea, which is the fusion of ML and Robotics techniques for vehicle trajectory prediction.

References

[1] Q. Yao, Y. Tian, Q. Wang, and S. Wang. Control strategies on path tracking for autonomous vehicle: State of the art and future challenges. *IEEE Access*, 8:161211–161222, 2020.

[2] Eduardo D Sontag. *Mathematical control theory: deterministic finite dimensional systems*, volume 6. Springer Science & Business Media, 2013.

[3] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, and Michael Young. Machine learning: The high interest credit card of technical debt. In *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*, 2014.

[4] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[5] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[7] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017.

[8] ByeoungDo Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 399–404. IEEE, 2017.

[9] Yuexin Ma, Xinge Zhu, Sibo Zhang, Ruigang Yang, Wenping Wang, and Dinesh Manocha. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6120–6127, 2019.

[10] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.

[11] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2090–2096. IEEE, 2019.

[12] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Nitin Singh, and Jeff Schneider. Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2095–2104, 2020.

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

[14] Hang Zhao, Jiyang Gao, Tian Lan, Chen Sun, Benjamin Sapp, Balakrishnan Varadarajan, Yue Shen, Yi Shen, Yuning Chai, Cordelia Schmid, et al. Tnt: Target-driven trajectory prediction. *arXiv preprint arXiv:2008.08294*, 2020.

[15] Yingfan Huang, Huikun Bi, Zhaoxin Li, Tianlu Mao, and Zhaoqi Wang. Stgat: Modeling spatial-temporal interactions for human trajectory prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6272–6281, 2019.

[16] Sergio Casas, Cole Gulino, Renjie Liao, and Raquel Urtasun. Spatially-aware graph neural networks for relational behavior forecasting from sensor data. *arXiv preprint arXiv:1910.08233*, 2019.

[17] Lidan Zhang, Qi She, and Ping Guo. Stochastic trajectory prediction with social graph network. *arXiv preprint arXiv:1907.10233*, 2019.

[18] Donsuk Lee, Yiming Gu, Jerrick Hoang, and Micol Marchetti-Bowick. Joint interaction and trajectory prediction for autonomous driving using graph neural networks. *arXiv preprint arXiv:1912.07882*, 2019.

[19] Abduallah Mohamed, Kun Qian, Mohamed Elhoseiny, and Christian Claudel. Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14424–14432, 2020.

[20] Sumit Kumar, Yiming Gu, Jerrick Hoang, Galen Clark Haynes, and Micol Marchetti-Bowick. Interaction-based trajectory prediction over a hybrid traffic graph. *arXiv preprint arXiv:2009.12916*, 2020.

[21] Sergio Casas, Wenjie Luo, and Raquel Urtasun. Intentnet: Learning to predict intention from raw sensor data. In *Conference on Robot Learning*, pages 947–956, 2018.

[22] Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Jeff Schneider, David Bradley, and Nemanja Djuric. Deep kinematic models for kinematically feasible vehicle trajectory predictions.

[23] Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. *arXiv preprint arXiv:1905.01296*, 2019.

[24] Karttikeya Mangalam, Harshayu Girase, Shreyas Agarwal, Kuan-Hui Lee, Ehsan Adeli, Jitendra Malik, and Adrien Gaidon. It is not the journey but the destination: Endpoint conditioned trajectory prediction. *arXiv preprint arXiv:2004.02025*, 2020.

[25] Lingyao Zhang, Po-Hsun Su, Jerrick Hoang, Galen Clark Haynes, and Micol Marchetti-Bowick. Map-adaptive goal-based trajectory prediction. *arXiv preprint arXiv:2009.04450*, 2020.

[26] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. MultiPath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. In *Conference on Robot Learning*, pages 86–99, 2020.

[27] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[28] SY Chen. Kalman filter for robot vision: a survey. *IEEE Transactions on Industrial Electronics*, 59(11):4409–4420, 2011.

[29] Thomas M Howard, Colin J Green, and Alonzo Kelly. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In *Field and Service Robotics*, pages 69–78. Springer, 2010.

[30] 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)*, pages 1094–1099. IEEE, 2015.

[31] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin. Combining optimal control and learning for visual navigation in novel environments. In *Conference on Robot Learning*, pages 420–429. PMLR, 2020.

[32] Tung Phan-Minh, Elena Corina Grigore, Freddy A Boulton, Oscar Beijbom, and Eric M Wolff. CoverNet: Multimodal behavior prediction using trajectory sets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14074–14083, 2020.

[33] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[34] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

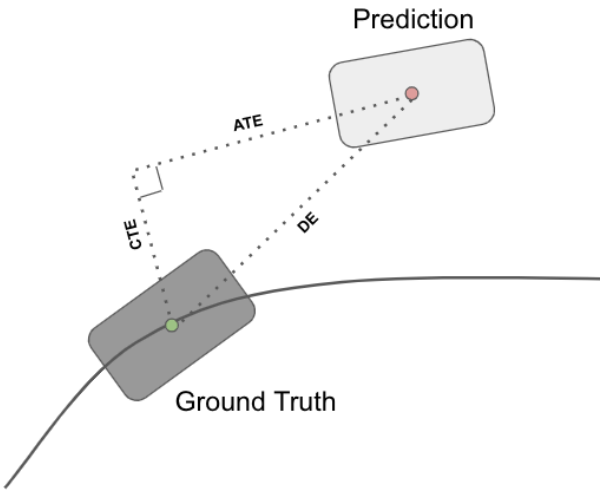


Figure 7: Illustration of the decomposition of displacement error (DE) into its along-track error (ATE) and cross-track error (CTE) components at a given timestep.

6 Appendix

6.1 Error Metrics

6.1.1 Average Displacement Error

We report the commonly used average displacement error (which we call avg DE, but is frequently abbreviated ADE) as a metric to quantify errors between predicted and ground-truth trajectories. This quantity is defined as the average ℓ_2 distance between the predicted and ground truth future trajectories across all timesteps for all actors.

6.1.2 Average Along Track Error and Cross Track Error

To further analyze prediction errors, we decompose errors into their along-track and cross-track components. These errors are calculated in the path-relative coordinate frame of the ground truth trajectory. We follow the methodology in [25] – given the ground truth trajectory, τ_{xy}, we re-sample τ_{xy} at a fixed spatial resolution of $\delta_\tau = 0.1$m, giving us the ground-truth path, ρ^*_{xy}. We then project the ground truth trajectory, τ_{xy}, and predicted trajectory, $\hat{\tau}_{xy}$, onto the ground truth path ρ^*_{xy}. The projection decomposes the error into the along-track and cross track components. Given the along-track and cross-track representations of the ground truth trajectory, τ_{ac}, and predicted trajectory, $\hat{\tau}_{ac}$, we calculate along-track error (ATE) and cross-track error (CTE) for a given timestep, t, as follows:

$$\textit{ATE} = \left|\hat{\tau}_a^t - \tau_a^t\right| \qquad \textit{CTE} = \left|\hat{\tau}_c^t\right| \tag{3}$$

We report the average along-track error (avg ATE) and average cross-track error (avg CTE) by averaging over all timesteps across all actors.