This is the Way: Differential Bayesian Filtering for Agile Trajectory Synthesis

Trent Weiss and Madhur Behl
Department of Computer Science
University of Virginia, Charlottesville, USA
{ttw2xk & madhur.behl}@virginia.edu

***Abstract*— One of the main challenges in autonomous racing is to design algorithms for motion planning at high speed, and across complex racing courses. End-to-end trajectory synthesis has been previously proposed where the trajectory for the ego vehicle is computed based on camera images from the racecar. This is done in a supervised learning setting using behavioral cloning techniques. In this paper, we address the limitations of behavioral cloning methods for trajectory synthesis by introducing Differential Bayesian Filtering (DBF), which uses probabilistic Bézier curves as a basis for inferring optimal autonomous racing trajectories based on Bayesian inference. We introduce a trajectory sampling mechanism and combine it with a filtering process which is able to push the car to its physical driving limits. The performance of DBF is evaluated on the DeepRacing Formula One simulation environment and compared with several other trajectory synthesis approaches as well as human driving performance. DBF achieves the fastest lap time, and the fastest speed, by pushing the racecar closer to its limits of control while always remaining inside track bounds.**

SUPPLEMENTARY VIDEO

This paper is accompanied by a narrated video of the performance: `https://youtu.be/5SezU0ICaug`

I. INTRODUCTION

In motorsport racing, there is a saying that *If everything seems under control, then you are not going fast enough*. Expert racing drivers have split second reaction times and routinely drive at the limits of control, traction, and agility of the racecar - under high-speed and close proximity situations. Autonomous racing presents unique opportunities and challenges in designing algorithms that can operate firmly on the limits of perception, planning, and control.

While autonomous vehicle research and development is focused on handling routine driving situations, achieving the safety benefits of autonomous vehicles also requires a focus on driving at the limits of the control of the vehicle. In recent years autonomous racing competitions, such as F1/10 autonomous racing [1], [2] and the Indy Autonomous Challenge [3] are becoming proving grounds for testing motion planning, and control algorithms at high speeds. With the autonomous racing application in mind, this paper focuses on the problem of trajectory synthesis or motion planning for an autonomous racecar. In our previous work [4], we have demonstrated end-to-end autonomous racing in the widely popular Formula One (F1) game used by real F1 drivers. Deep learning based approaches for trajectory synthesis for autonomous vehicles have either been one where a trajectory is predicted via a prediction of future waypoints for the ego vehicle to follow, or in some cases a parameterized curve is predicted instead and waypoints are sampled from the curve [5]–[7]. In either case, these techniques still rely on supervised learning from expert behavior for predicting the trajectory of the ego vehicle. Once the predicted trajectory is computed a low-level controller such as pure-pursuit or a model predictive control can follow that trajectory.

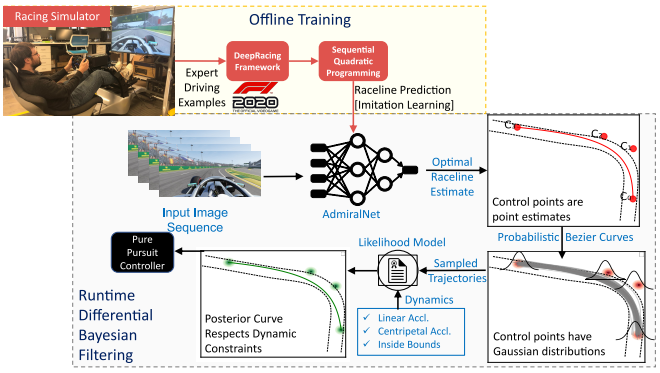

Fig. 1. The key idea in Differential Bayesian Filtering is that we fit Gaussian distributions over a Bézier curve's control points to generate a distribution of trajectories that respect the ego vehicle's dynamic limits.

However, behavior cloning based trajectory synthesis is brittle [8] because at best it can generate trajectories which are averaged from the input/training data. These methods do not generalize to computing trajectories which respect vehicle dynamics, or track bounds at all times. Nor can they leverage the knowledge of vehicle dynamics.

In this paper, we address major limitations of supervised learning methods for trajectory synthesis for autonomous racing by introducing a new method called Differential Bayesian Filtering (Fig. 1). This is a significant improvement over previous methods for trajectory synthesis, and can be summed up with 3 major contributions:

1) The ability to learn distributions over the space of desired trajectories.
2) A Monte-Carlo sampling method for inferring closer-to-optimal trajectories from those distributions
3) A filtering method to combine deep learning trajectory synthesis with knowledge of the vehicle's feasibility limits.

We show empirically in Section VII that our novel frame-

work, built on top of Probabilistic Bèzier Curves and Monto-Carlo based Bayesian inference, generates trajectories which are outside of the training data distribution and results in significantly better performance than the best human driving example in the training data.We select autonomous racing as the application domain for our work, but it can generalize to motion planing for any autonomous vehicle.

II. Related Work

There is existing work on trajectory synthesis for autonomous driving [9]–[11]. However, these works only consider a "0^{th}" order view of the problem i.e. planning a trajectory based only on the ego vehicle's position. They do not consider any differential or higher order constraints which are desirable for speed and acceleration behavior.

In [12], authors present a probabilistic approach for trajectory synthesis which predicts a 4^{th} order spline. However, [12] does not consider distributions over the space of polynomials, as we do, but uses Gaussians over fixed points along a predicted polynomial as a convenient means of constructed their network's loss function. Additionally, [12] only uses behavioral cloning which is not optimal and does not align with the goal of pushing the vehicle to its limits.

Researchers have looked at the problem of determining an optimal raceline [13], [14]. In these works, the raceline is determined by an offline optimization that can incorporate knowledge of vehicle dynamics. Our method, on the other hand, is *online* and infers optimal behavior from desirable characteristics of the vehicle's motion at runtime. We aren't proposing a new method to find the optimal raceline for a track, rather we are proposing a runtime method, that generates fast trajectories for a racecar - which converges to the optimal raceline when the racecar runs by itself but can also be extended to multi-agent racing situations.

In [15]–[17], the authors use fully end-to-end style architectures. I.e. images from the driver's point of view are mapped directly to control outputs for the car (steering and throttle). However, this approach is very brittle and usually does not generalize well to out-of-distribution images. Other work [18], [19] remedies this limitation by using neural networks to predict a series of waypoints for the autonomous vehicle to follow. Waymo's ChauffeurNet [20] learns to follow specified waypoints and specified speeds. The survey in [21] also lists several end-to-end architectures for autonomous driving. Most use some form of Convolutional Neural Networks, sometimes integrated with LSTM cells or another form of recurrent network. For example, in [22], the authors utilize a CNN-LSTM architecture for predicting a fixed number of waypoints and velocities for the car to follow, similiar to [20]. However, predicting sufficient waypoints leads to the curse of dimensionality as the neural network needs to predict proportionally more parameters to specify more waypoints. In our method, we only need to predict the control points of a parameterized Bezier curve, which fixes the dimensionality of our problem and allows implicit encoding of the predicted trajectories derivatives. [23] is an example of work which uses Deep Reinforcement Learning (DRL) to autonomously race in the Gran Turismo game. This is close to our work in terms of the problem setting but very different in terms of methodology. DRL techniques require numerous exploration runs of the action space, including experiencing many crashes while learning to maximize racing related rewards. Our method requires only a few hours worth of training data and is able to push the car to its limits. Additionally, our Formula One DeepRacing framework does not require any special access to the internal state of the game engine - it is reproducible by anyone who owns the game unlike the Grand Turismo work which required changing game state to enable DRL.

We build upon our previous work in [24], [25] that uses a neural network to predict a Bèzier curve as a canonical representation of a desired trajectory for the autonomous vehicle. This model was shown to outperform waypoint prediction, and the parameterized representation is a convenient form for applying Bayesian inference techniques to estimate a more-optimal trajectory. We next present a brief overview of probabilistic Bézier curves, an extension of Bèzier curves to a Gaussian probability model, which will form the basis of our Bayesian method for agile trajectory synthesis.

III. Probabilistic Bèzier Curves

A Bézier curve is a parametric curve used in computer graphics and related fields. The curve, a linear combination of Bernstein polynomials, is named after Pierre Bézier, who developed them to model Renault racecars. A Bézier curve is formed from a combination of Bernstein polynomials (Eq 1) that maps a scalar parameter $s \in [0, 1]$ to a point in a euclidean space of dimension d, R^d. More specifically, a Bézier curve is a polynomial combination of a set of "control points". A Bézier curve of degree k isdefined by k+1 control points: $\{\mathbf{C}_0, \mathbf{C}_1, \mathbf{C}_2, ..., \mathbf{C}_k \in R^d\}$. The corresponding Bézier curve, $\mathbf{B} : [0, 1] \rightarrow R^d$, as a function of a unitless scalar, s, is:

$$b_{i,k}(s) = \binom{k}{i}(1-s)^{k-i}s^i \tag{1}$$

$$\mathbf{B}(s) = \sum_{i=0}^{k} b_{i,k}(s)\mathbf{C}_i \tag{2}$$

Note that a Bézier curve always starts at $\mathbf{C}_0$ ($s = 0$) and ends at $\mathbf{C}_k$ ($s = 1$). A Bézier curve's derivative w.r.t s is:

$$\frac{d\mathbf{B}}{ds} = k\sum_{i=0}^{k-1} b_{i,k-1}(s)(\mathbf{C}_{i+1} - \mathbf{C}_i) \tag{3}$$

Under this formulation, the scalar parameter s is just unitless parameter on $[0, 1]$. To express the curve as a function of *time*, we specify a total amount of time, Δt, for the curve to go from $\mathbf{C}_0$ to $\mathbf{C}_k$. I.e. time, t, can be expressed as $t = s\Delta t$. Equivalently, $s = \frac{t}{\Delta t}$. Under this formulation, the velocity vector, w.r.t time, of a Bézier curve is simply:

$$\frac{ds}{dt} = \frac{1}{\Delta t} \tag{4}$$

$$\frac{d\mathbf{B}}{dt} = \frac{ds}{dt}\frac{d\mathbf{B}}{ds} = \frac{1}{\Delta t}\frac{d\mathbf{B}}{ds} \tag{5}$$

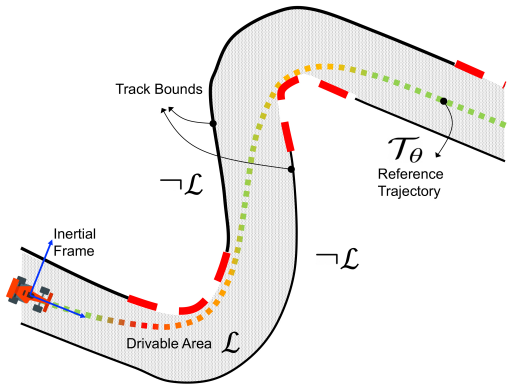

Fig. 2. High-level view of autonomous racing. The driveable and undrivable areas are $\mathcal{L}$ and $\neg\mathcal{L}$, respectively. We present a Bayesian approach to determining an optimal trajectory, $\mathcal{T}_\theta$, for the ego vehicle to follow.

Δt determines how fast the curve is as a function of time, smaller Δt implies a faster curve and a larger Δt implies a slower curve. How this chosen constant determines the velocity of the curve will be important for our Bayesian formulation of path planning.

Hug et al. [26] introduced the concept of probabilistic Bézier curves. A probabilistic curve is defined not by a fixed set of control points, but by a set of mutually-independent Gaussian distributions over each control point, where each Gaussian vector is defined by a mean, $\mathbf{C}_i \in R^d$, and a covariance matrix, $\mathbf{\Sigma}_i \in R^{d\times d}$. I.e. a probabilistic Bézier curve, $\mathcal{B}(s)$, is defined by:

$$\{\mathcal{C}_0 \sim \mathcal{N}(\mathbf{C}_0, \mathbf{\Sigma}_0),\ \ldots\ ,\ \mathcal{C}_k \sim \mathcal{N}(\mathbf{C}_k, \mathbf{\Sigma}_k)\} \tag{6}$$

$$\mathcal{B}(s) \sim \sum_{i=0}^{k} b_{i,k}(s)\mathcal{C}_i \tag{7}$$

Where each $\mathcal{C}_i \sim \mathcal{N}(\mathbf{C}_i, \mathbf{\Sigma}_i)$ is independent of the others.

IV. Problem Formulation

This work is focused on a Bayesian interpretation of trajectory synthesis for autonomous racing. Given some sensor inputs, an autonomous racing agent needs to generate a trajectory to follow that represents desirable racing behavior, where "desirable" means a trajectory that is as fast as possible without violating any kinematic/dynamic constraints of the vehicle or going outside the boundaries of the track. In this work, a *trajectory*, which we denote as $\mathcal{T}$, is a smooth curve in the ego vehicle's task space: R^2. $\mathcal{T}$ is a set of points that can be expressed as a C^∞ function of time, such that the terms velocity and acceleration have their intuitive meaning: the first and second time derivatives of the curve, respectively. For this work, we focus on trajectories that can be fully described by a set of parameters: θ. We denote such a parameterized trajectory as $\mathcal{T}_\theta$. For this work, we take θ to be the control points of a Bèzier Curve. Figure 2 gives a high-level overview of this problem setting. An autonomous racing agent needs to select a parameterized trajectory that is as fast as possible without leaving the drivable area or exceeding the car's physical limits. Once a trajectory is selected, a classical path-following algorithm can be used to decide on steering and throttle commands for the ego vehicle. In this work, we select a Pure Pursuit controller [27], but other path-following algorithms (such as a Model Predictive Controller) could be used - the choice is agnostic to our framework. The optimal racing line (ORL), $\mathcal{T}^*$, is the trajectory with maximal average velocity (over time, t) that is both achievable under the car's dynamic limits and is wholly contained within the track boundaries. Let $\Lambda(\mathcal{T})$ be a boolean function that is true iff $\mathcal{T}$ is physically achievable. We denote the region within the track bounds with the symbol $\mathcal{L}$. If $\dot{\mathcal{T}}$ and $|\dot{\mathcal{T}}|$ represent the velocity and speed, respectively, of $\mathcal{T}$, the optimal racing line is:

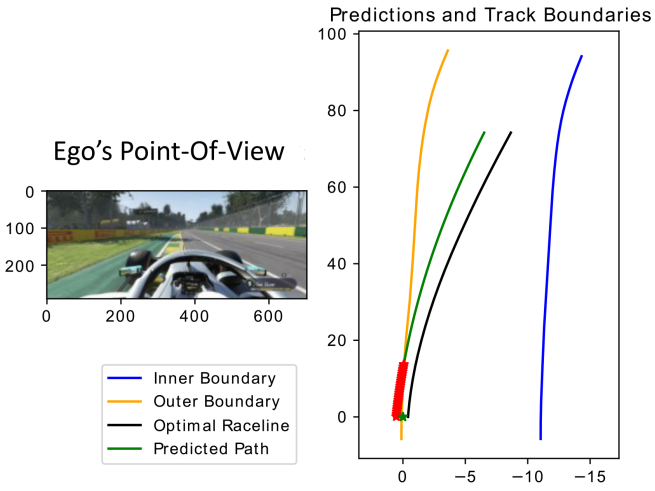

Fig. 3. A neural network is trained the predict $\mathcal{T}^*$ (in black) 2.25[s] into the future based on images from the driver's point-of-view. However, it's predictions (in green) can be incorrect. We show that our Bayesian framework can mitigate the effect of incorrect network predictions and produce more optimal behavior.

$$\mathcal{T}^* = \text{argmax}(\mathcal{T})\ E_t[|\dot{\mathcal{T}}|] : (\mathcal{T} \subset \mathcal{L})\ \wedge \Lambda(\mathcal{T}) \tag{8}$$

We present a method of Bayesian inference, called Differential Bayesian Filtering, to estimate $\mathcal{T}^*$, which can then be passed to a path-following algorithm. This technique is an extension of our previous work in [24], [25] that uses a sequence of images as the input to a neural network, called AdmiralNet, that was trained to predict the control points of a Bèzier curve approximation to $\mathcal{T}^*$. For details on AdmiralNet's architecture, we refer the reader to [24]. However, our work in [24] only provides a deterministic Bèzier curve that might be incorrect for images that the neural network has never seen before (Fig. 3). To address this limitation, we use Bayesian inference for improving the point estimate by taking the output of that neural network as the mean of a probabilistic Bèzier curve to serve as a Gaussian prior distribution for Bayesian inference.

V. Raceline Estimation

We utilize Sequential Quadratic Programming (SQP) to specify the optimal racing line that the neural network is trained to predict. To start, we select the minimum curvature path that completes the entirety of the racetrack and stays

within the track boundaries, a set of points, P:

$$P = [p_0, p_1, ..., p_{N-1}] \overset{argmin}{} \sum_{i=0}^{N-1} \kappa_i \tag{9}$$

Where $[p_0, p_1, ..., p_{N-1}] \in R^2$ traverses the entirety of the racetrack and κ_i is the curvature of the path at p_i. However, this is just a set of points in R^2. To specify the optimal raceline as a trajectory (a function of time), we utilize SQP optimization to assign a speed at each point on the minimum curvature path, from which a time at each point readily follows. We uniformly sample points at 1.5m intervals from the minimum curvature path , such that v_i is the desired speed along the path at the point p_i. We then impose two inequality constraints on the square of the car's speed (v^2) at each point:

First, we limit the centripetal (lateral) acceleration, a_c, of the car to 26.5 $[\mathrm{m/s^2}]$ ($\sim$2.7G). Real F1 cars experience $\sim 3-4$G of centripetal acceleration, so we select this limit as a conservative estimate.

$$\frac{{v^2}_i}{R_i} \leq 26.5[\mathrm{m/s^2}] \ \forall i \tag{10}$$

Where $R_i = \frac{1}{\kappa_i}$ is the radius of curvature of the path at p_i. Second, we limit the longitudinal acceleration, a_l, of the car by assuming the car's longitudinal acceleration is constant between each p_i. Let Δr represent the path-length between any two adjacent points on the path and a_l represent longitudinal (forward) acceleration. This constraint is then:

$${v_i}^2 = {v_{i-1}}^2 + 2a_{l_{i-1}}\Delta r \tag{11}$$

Because we discretize the path at intervals of $\Delta r = 1.5[\mathrm{m}]$:

$$a_{min}(v_{i-1}) \leq \frac{{v_i}^2 - {v_{i-1}}^2}{3} \leq a_{max}(v_{i-1}) \tag{12}$$

a_{max} is a function that maps the car's speed to the maximum forward acceleration the car can achieve at that speed, which decreases as the car goes faster due to drag. a_{min} is a function that maps the car's speed to the fastest braking rate the car can achieve at that speed (a negative sign here indicates braking). This will also decrease (braking is negative acceleration) as the car goes faster, because drag helps the car slow down more at higher speeds. The closed form of a_{max} and a_{min} is very complex, as they both involve the complex aerodynamics of an F1 car body. For this work, we determine a_{min} and a_{max} with step-response test on the car in the DeepRacing testbed and measured the maximum achievable acceleration and braking as a function of speed. We use linear interpolation on these data as a_{max} and a_{min}.

Setting optimal velocities at each point then becomes a vector-space optimization problem. Given a vector $\mathbf{v} = [{v^2}_0, {v^2}_1, ..., {v^2}_{N-1}]$, we determine velocities on the minimum-curvature path to be the solution to:

$$max \sum_{i=0}^{N-1} \mathbf{v}_i \ s.t. \ \ \mathbf{Av} \leq \boldsymbol{\gamma}_1, \mathbf{Bv} \leq \boldsymbol{\gamma}_2, \mathbf{Bv} \geq \boldsymbol{\gamma}_3 \tag{13}$$

$$\mathbf{A} = \mathrm{diag}([\frac{1}{R_0}, \frac{1}{R_1}, ..., \frac{1}{R_{N-1}}]) \tag{14}$$

$$\mathbf{B} = \begin{bmatrix} \frac{-1}{3} & \frac{1}{3} & 0 & ... & 0 \\ 0 & \frac{-1}{3} & \frac{1}{3} & ... & 0 \\ & & ... & & \\ 0 & 0 & ... & \frac{-1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & ... & 0 & \frac{-1}{3} \end{bmatrix}$$

$$\boldsymbol{\gamma}_1 = \begin{bmatrix} 26.5 \\ 26.5 \\ ... \\ 26.5 \\ 26.5 \end{bmatrix} \quad \boldsymbol{\gamma}_2 = \begin{bmatrix} a_{max}(v_0) \\ a_{max}(v_1) \\ ... \\ a_{max}(v_{N-2}) \\ a_{max}(v_{N-1}) \end{bmatrix} \quad \boldsymbol{\gamma}_3 = \begin{bmatrix} a_{min}(v_0) \\ a_{min}(v_1) \\ ... \\ a_{min}(v_{N-2}) \\ a_{min}(v_{N-1}) \end{bmatrix}$$

We then train our neural network architecture to predict this raceline, $\mathcal{T}^*$, 2.25[s] into the future. 2.25[s] was selected as the duration of a typical braking zone for an F1 car. The input to this neural network is a sequence of images from the ego vehicle's point of view. Its output is a Bézier curve approximation of $\mathcal{T}^*$, starting at a point on the raceline closest to the ego vehicle, as seen in Figure 3. However, since this output is generated using supervised learning, it may not be correct as the network may encounter input image sequences which are outside of the training distribution. This network prediction serves to specify a prior distribution for our Bayesian filtering framework.

VI. Differential Bayesian Filtering

Since the *differential* properties of $\mathcal{T}^*$ are well-defined, they can serve as the basis for inferring the optimal raceline given a prior. Specifically, for the optimal raceline:

1) $E[|\dot{\mathcal{T}}^*|]$ (Eq. 8) is maximized.
2) $\mathcal{T}^*$ respects the vehicle's dynamic constraints.
3) The signed distance from $\mathcal{T}^*$ to the track bounds is strictly non-positive. "Signed distance" means euclidean distance to the track boundary, but with a negative sign for points inside the track boundaries.

The goal of Differential Bayesian Filtering is to infer a Bèzier curve approximation of $\mathcal{T}^*$ by performing Bayesian estimation on $\mathcal{T}^*$'s differential properties, even if the global properties of $\mathcal{T}^*$ are not known. In the context of our problem, this means that the resulting curve should have a higher average velocity, but still be within the track bounds and be physically achievable by the ego vehicle.

Recall that we take the control points of Bèzier curve to be the parameters, θ, of our estimate of the optimal raceline: $\mathcal{T}_\theta$. I.e. $\theta = \{\mathbf{C}_0, \mathbf{C}_1, ..., \mathbf{C}_{k-1}\}$ as described in section III. We use a probabilistic Bèzier curve, Equation 6 from Section III, as a prior distribution, $p(\theta)$, and apply recursive Bayesian estimation with a likelihood function derived from $\mathcal{T}^*$'s differential properties to infer a more accurate estimate of true optimal behavior. In general, for any curve that can be parameterized by θ, Bayes' Theorem says:

$$p(\theta|x) = \frac{l(\theta|x)p(\theta)}{\int l(\theta|x)p(\theta)d\theta} \tag{15}$$

The evidence, x, is sensor data or other state information that is available to the autonomous agent. For this work, we take this evidence to be the following:

1) A permissible offset distance from the track boundaries
2) The maximum allowed longitudinal acceleration.
3) The maximum allowed centripetal acceleration.

$l(\theta|x)$ is a function of θ and x that represents the likelihood that $\mathcal{T}_\theta$ (the trajectory defined by θ) is the optimal raceline given the evidence, x. The prior, $p(\theta)$, and likelihood function, $l(\theta|x)$, are the defining features of a Differential Bayesian Filtering model. Intuitively, this likelihood should be selected such that curves with *desirable* differential properties have a higher likelihood and curves with *undesirable* differential properties should have a lower likelihood, such that the posterior distribution, $p(\theta|x)$, represents a more accurate belief about $\mathcal{T}^*$. Other desirable racecar behaviors such as collision-free trajectories for multi-agent racing could be incorporated as another likelihood function in the future.

A. Gaussian Prior on $\mathcal{T}^*$

For this work, we use a Probabilistic Bezier Curve as the basis for a prior distribution on the parameters of the optimal racing line. We take the point-estimate output of our neural network model in [24] as the mean of our prior distribution and with identity covariance. I.e. our prior $p(\theta)$ is as follows:

$$\{\mathcal{N}(\hat{\mathbf{C}}_0, \begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix}),\ ...\ \mathcal{N}(\hat{\mathbf{C}}_k, \begin{bmatrix}1.0 & 0\\ 0 & 1.0\end{bmatrix})\} \tag{16}$$

Where $\{\hat{\mathbf{C}}_0\ ...\ \hat{\mathbf{C}}_k\}$ are the point-estimate control points of the curve predicted by our neural network. Identity covariance is chosen for this initial work for simplicity, but other priors could be specified, e.g. where the covariance is determined based on the trajectories from the training data.

B. Likelihood Specification for $\mathcal{T}^*$

We want to specify our Bayesian model to make physically infeasible or out-of-bounds curves less likely and feasible curves that remain in-bounds more likely. To achieve this goal, we utilize Bayes' Theorem where the likelihood function is a function of the *differential properties* of the trajectory defined by θ, rather than a function of only θ:

$$p(\theta|x) = \frac{l(\mathcal{T}_\theta, \dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta|x)p(\theta)}{\int l(\mathcal{T}_\theta, \dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta|x)p(\theta)d\theta} \tag{17}$$

Note that under this formulation, there need not be a direct connection between the evidence, x, and the true parameters of $\mathcal{T}^*$. In our case, the likelihood is a function of the derivatives of $\mathcal{T}^*$. To formally specify this likelihood, let a_c and a_l represent centripetal and longitudinal acceleration, respectively. Let $d(\mathcal{T}_\theta)$ be the maximum signed distance from $\mathcal{T}_\theta$ to the track boundaries. In this context, "signed distance" is Euclidean distance but with a negative sign for points on $\mathcal{T}_\theta$ that are inside the track bounds and a positive sign for points outside the bounds. We define acceleration likelihoods (Eqs. 18, 19) and a boundary likelihood (Eq. 20) as follows:

$$l_1(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) = \begin{cases} e^{-\beta_1 \Delta a_c(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta)}, & \Delta a_c(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) > 0 \\ 1, & \text{otherwise} \end{cases} \tag{18}$$

$$l_2(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) = \begin{cases} e^{-\beta_2 \Delta a_l(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta)}, & \Delta a_l(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) > 0 \\ 1, & \text{otherwise} \end{cases} \tag{19}$$

$$l_3(\mathcal{T}_\theta \subset \mathcal{L}) = \begin{cases} e^{-\beta_3 (d(\mathcal{T}_\theta) - d_{min})}, & d > d_{min} \\ 1, & \text{otherwise} \end{cases} \tag{20}$$

In this context, $\Delta a_c(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta)$ represent how much $\mathcal{T}_\theta$ violates the centripetal acceleration limits of the car; $\Delta a_c(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) > 0$ implies the curve is too aggressive (too much centripetal acceleration) and $\Delta a_c(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) <= 0$ implies the curve is within the limit. Δa_l represents the same function, but for longitudinal acceleration (braking/throttling). These deltas refer to the same feasibility limits (linear and centripetal acceleration) that were used when determining the optimal raceline for training the neural network. We refer to trajectories that exceed these acceleration limits as infeasible trajectories and to trajectories within the limits as feasible trajectories. β_1, β_2, β_3, and d_{min} are all tuneable hyperparameters. These parameters represent how strongly each factor of the likelihood function should be weighted. Our likelihood function for Bayesian inference is the product of these three factors:

$$l(\mathcal{T}_\theta, \dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta|x) = l_1(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) l_2(\dot{\mathcal{T}}_\theta, \ddot{\mathcal{T}}_\theta) l_3(\mathcal{T}_\theta \subset \mathcal{L}) \tag{21}$$

The core idea is that, given a prior distribution that is at or beyond the limits of control, our likelihood model infers optimal behavior by encouraging feasible trajectories that are within bounds and discouraging infeasible or out-of-bounds trajectories. Our method is not limited to these definitions of feasibility or to these specific acceleration limits, they are just a simplifying choice. More complex vehicle dynamics could also be included in our approach by specifying an appropriate likelihood function based on such a model, to account for slip angles, tire forces etc.

C. Sampling-Based Differential Bayesian Filtering

The closed-form solution to the posterior distribution under our formulation is not readily obvious. To remedy this, we now describe a sampling-based approach for generating samples from the posterior distribution without the need to solve for it analytically. We employ Monte Carlo-based sampling from the posterior distribution over θ given the likelihood specified in subsection VI-B and the prior from subsection VI-A, $p(\theta)$. We draw a sample of N curves $p(\theta)$: $[\theta_1, \theta_2, \theta_3, ..., \theta_N]$ from $p(\theta)$. These N curves are then assigned a weight based on equations 18, 19, and 20. This approach is extensible to more sophisticated sampling schemes such as Adaptive Monte Carlo, but a fixed number of samples is used in this work. We then employ a re-sampling step to infer a posterior distribution. Each sample θ_i is assigned a weighting factor γ_i according to:

$$\alpha(\theta_i) = l_1(\dot{\mathcal{T}}_{\theta_i}, \ddot{\mathcal{T}}_{\theta_i}) l_2(\dot{\mathcal{T}}_{\theta_i}, \ddot{\mathcal{T}}_{\theta_i}) l_3(\mathcal{T}_{\theta_i} \subset \mathcal{L}) \tag{22}$$

$$\gamma(\theta_i) = \frac{\alpha(\theta_i)}{\sum_{i=1}^{N} \alpha(\theta_i)} \tag{23}$$

l_1, l_2, and l_3 refer to equations 18, 19, and 20, respectively. The mean of the posterior distribution is then taken as

$\sum_{i=1}^{N} \gamma_i * \theta_i$. This process repeats for a desired number of iterations. Algorithm 1 describes the high-level loop of our approach. Figure 4 depicts our approach graphically. After sampling the grey curves and assigning their weights, γ_i, the posterior mean (in green) is inside the track bounds and is dynamically feasible while maintaining a high speed.

Novelty of DBF compared to Behavioral Cloning approaches: The novelty of the DBF method is the following: (1) Instead of tuning a neural network for predicting a single parameterized trajectory as one would in a behavioral cloning setting, we view the problem as one of learning probability distributions over the parameters of the optimal trajectory; (2) At runtime, we sample from the learned distributions to create candidate trajectories; (3) Using our likelihood specification, we emphasize samples with desirable properties and discourage samples with undesirable properties; (4) Finally, we use a weighting factor based on this likelihood to generate a posterior distribution that exhibits more optimal behavior. This approach differs from the typical view of a machine-learned trajectory synthesis model by incorporating knowledge of the vehicle's dynamics to improve the point-estimate produced by a deep model.

Algorithm 1: Differential Bayesian Filtering

DBF()

$p(\theta) \leftarrow \mathcal{N}(AdmiralNet(), \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix})$

while $\neg$ *terminate()* **do**

//Draw N samples from prior (each sample is //control points for a Bèzier Curve)

$\{\theta_1, \theta_2, ...\theta_N\} \leftarrow sample(\text{p}(\theta)\})$

//Compute derivatives for each sampled curve

$\mathcal{T}_{\theta_i} \leftarrow$ Bèzier Curve defined by θ_i

$\dot{\mathcal{T}}_{\theta_i} \leftarrow$ Velocity of $\mathcal{T}_{\theta_i}$

$\ddot{\mathcal{T}}_{\theta_i} \leftarrow$ Acceleration of $\mathcal{T}_{\theta_i}$

//Compute weighting factors based on equation 23

$\gamma_i \leftarrow \gamma(\mathcal{T}_{\theta_i}, \dot{\mathcal{T}}_{\theta_i}, \ddot{\mathcal{T}}_{\theta_i})$

//Posterior is weighted sum of sampled curves

$\bar{\theta} = \sum_{i=1}^{N} \gamma_i * \theta_i$

$p(\theta) \leftarrow \mathcal{N}(\bar{\theta}, \begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix})$

return $p(\theta)$

VII. Experiments & Results

We now present an empirical evaluation of Differential Bayesian Filtering in a high-speed, Formula One™ racing environment. We use the neural network architecture from our previous work in [24] to provide a point estimate of a Bèzier Curve approximation to the optimal racing line, $\mathcal{T}^*$, given a sequence of images taken from the driver's point-of-view, as shown in Figure 3. This point estimate serves as the mean of our prior distribution, but with one key change. We scale the time delta, Δt, that the neural network was trained to predict by a factor of $\frac{1}{1.15} \approx .870$. I.e. we scale up the velocities of the prior by a factor of 15%. Because centripetal acceleration is proportional to the square

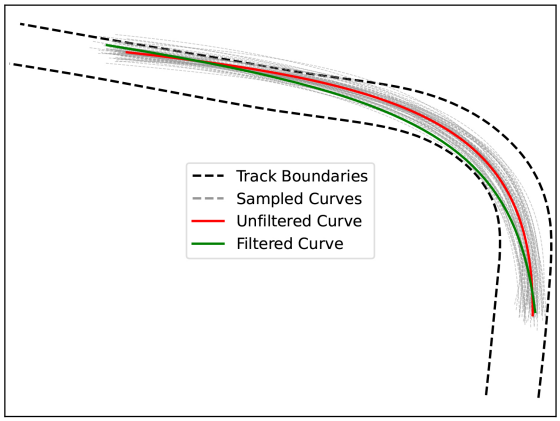

Fig. 4. One step of the Differential Bayesian Filtering algorithm. The prior (unfiltered) curve in red is infeasible, with too much centripetal acceleration. Our filtering approach produces a posterior (in green) that is closer to optimal with a softer arch that still maintains it's speed.

of velocity, this would increase the centripetal acceleration of the prior by a factor of $1.15^2 \approx 1.323$. With no other changes to the trajectory, this would push the car beyond its feasibility limits (we show this empirically in subsection VII-A).Even though this scaled trajectory is infeasible, we use this as a prior - meaning that the mean of the distribution is dynamically infeasible and will cause the racecar to spin out. This unstable prior is chosen in order to encourage the Bayesian sampling to produce candidate trajectories which are a mix of both stable and unstable raceline estimates. Consequentially, this increases the probability of finding a trajectory that is close to the vehicle's limits, but does not exceed them.Differential Bayesian Filtering improves this initial unstable prior into a more optimal Bèzier curve that is both dynamically feasible and faster than the neural networks point estimate. The resulting optimal Bézier curve is then passed to a Pure Pursuit [27] controller to generate steering and throttle commands for the autonomous racing agent. This Pure Pursuit controller uses an adaptive lookahead based on a control law of $l_d = 0.4v$ with l_d and v representing lookahead distance and the car's current speed, respectively. We evaluate the unfiltered approach, where just the point estimate off the neural network (with no velocity scaling) is passed straight to Pure Pursuit, as well as a filtered model with Differential Bayesian Filtering on the unstable prior to produce a posterior that is within the car's limits.

In addition, we also compare our approach to two other models that uses a behavioral cloning approach to mimic an expert human driver's behavior instead of following the optimal racing line. One of these models was trained to predict a Bèzier Curve that mimics the trajectory taken by the expert human driver. The other is closer to Waymo's ChauffeurNet [20] and predicts a series of waypoints followed by the expert example. We also include performance results when the Pure Pursuit controller is given ground-truth knowledge of the SQP-generated raceline the network was trained on.

Each model is run for 5 laps in our DeepRacing framework, using Codemaster's© F1™ 2019 racing video game and the DeepRacing closed-loop autonomous racing infrastructure [4]. On each run, we measure the following metrics:

TABLE I

HYPERPARAMETERS USED FOR OUR EXPERIMENTS

Hyperparameter	Value
β_1 (Eqn 18)	1.75
β_2 (Eqn 19)	2.5
β_3 (Eqn 20)	3.5
d_{min} (Eqn 20)	-0.875 meters
N (# of samples in VI-C)	250

1) Overall Lap Time
2) Average Speed
3) How many times the autonomous agent went outside the track bounds, what we call a "Boundary Failure" - Such a failure is defined as when ≥ 3 of the vehicle's tires go out of bounds.

In these experiments, order 7 curves were chosen (k=7) based on heuristics. Table I shows the values we use for all of the hyperparameters for our approach defined in section VI. Table II summarizes the performances of DBF in the DeepRacing simulator (means across 5 laps). DBF has the fastest overall lap time and the fastest average speed. This implies our method is taking a more efficient racing line and is doing so more aggressively without exceeding dynamic limits of the car. It is also worth noting that our method has no boundary failures. Our approach also improves lap time by $\sim 4.458[\text{s}]$ on average from the unfiltered neural network's raceline predictions. In the world of high-speed racing, where winners/losers can be decided by millisecond time differences, this is a very significant improvement. We also achieve lap times superior to the human driving examples used to train the neural network by $\sim 2.4[\text{s}]$. I.e. our technique pushes the vehicle faster than even the fastest example in the neural networks training distribution. All of our experiments were conducted on a PC with 12 CPU cores and an NVIDIA GTX1080Ti GPU. Each iteration of the filtering loop (250 samples) required an average of $0.034[\text{s}]$ of computation time, implying a rate of $\sim 29[\text{Hz}]$ for each iteration of DBF. Our overall algorithm, including a call to AdmiralNet and the pure pursuit controller, runs at $10[\text{Hz}]$.

A. Dynamics Analysis

We also present evidence that our method is achieving our stated goal, pushing the vehicle its limits. The Turn 3/Turn 4 chicane of Albert Park Circuit is a particularly challenging turn. It requires braking from almost max speed and navigate an S-shaped chicane. Fig 5 shows this chicane and how DBF (green curve) maintains speed through the turn by utilizing the available track width. Fig 6 shows the centripetal acceleration the car experiences in this chicane while controlled by each of 3 trajectory planners:

1) The unfiltered neural network predictions
2) The artificially accelerated (unstable) prior distribution
3) Our Differential Bayesian Filtering method
4) The best human example lap from our training set

The horizontal line shows the dynamic limit we assigned to the car. Note that when following the neural network's unfiltered predictions (dashed blue in Figure 5), the car drives too passively. There is significant room for the vehicle to drive faster without exceeding the vehicle's dynamic limits. Also note that when pushed faster artificially (as with our selected prior, dotted red in Figure 5), the car sees far too much centripetal acceleration (see Figure 6, even human drivers don't push the car that hard). The vehicle becomes unstable and spins out halfway through the chicane under these conditions. Our method results in the best of both worlds. The vehicle operates right at its dynamic limits, but does not exceed them. The vehicle carries more speed through the turns, but with an acceptable level of centripetal acceleration. This manifests as overall faster lap times.

In sum, we show that Differential Bayesian Filtering can produce a higher-performance racing agent that drives more aggressively (better lap time & speed) and more safely (not going out-of-bounds) than only following point estimates of the optimal racing line produced by a convolutional neural network. This technique is also generalizable to more application-specific likelihood models and prior distributions.

TABLE II

RESULTS FROM EXPERIMENTS IN THE F1 SIMULATOR

Model Configuration	Lap Time [s]	Overall Speed [m/s]	Number of Boundary Failures
Waypoint Prediction (Behavioral Cloning)	106.683	49.245	5.6
Bézier Curve Prediction (Behavioral Cloning)	101.72	52.193	1.8
Unfiltered Raceline Prediction	91.219	58.109	0
Fastest Human Lap (Training Data)	89.177	59.439	1
Ground-Truth Raceline	88.046	60.186	0
Raceline Prediction w/ Differential Bayesian Filtering (Ours)	**86.761**	**61.092**	**0**
Nicholas Latifi (in real-life 2019 Australian Grand Prix)	86.067	Unknown	0
Lewis Hamilton (in real-life 2019 Australian Grand Prix)	80.486	Unknown	0

Differential Bayesian Filtering outperforms the other methods and the best human lap in the training data by a significant margin. Our method is also very close to competing with real F1 drivers.

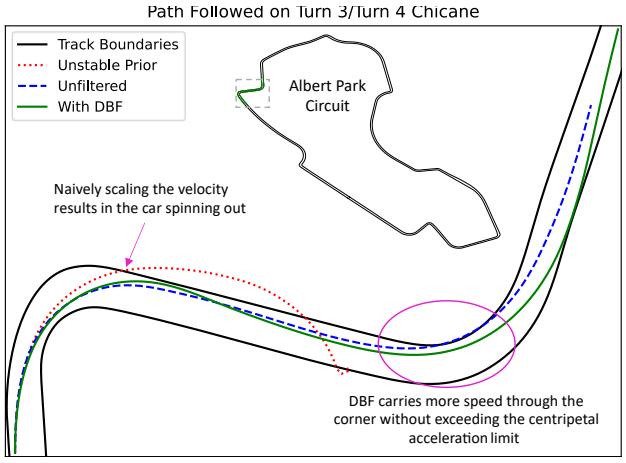

Fig. 5. Turn3/Turn4 Chicane of Albert Park Circuit. Note that our methods results in a faster path (green) that carries more speed through the turn. The unstable prior (red) is not physically achievable and results in a crash.

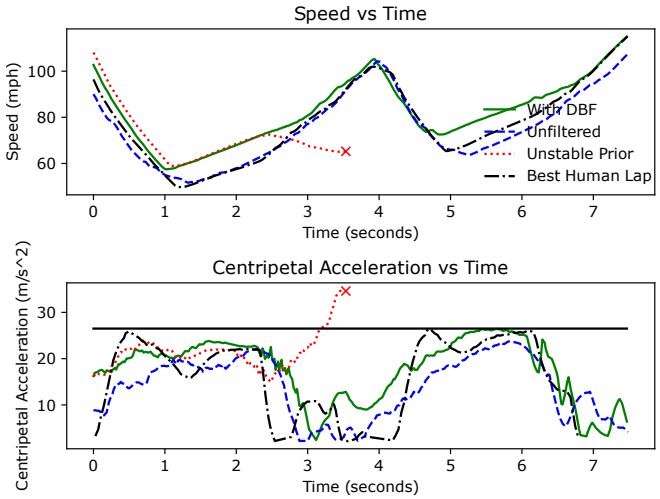

Fig. 6. Our artificially accelerated prior is dynamically infeasible and causes the car to spin out on the Turn3-Turn4 chicane. Differential Bayesian Filtering brings that prior back within the car's physical limits. Our specified dynamic limit of $26.5[\mathrm{m/s^2}]$ is marked on the lower plot. Our approach pushes the car right up to its dynamic limit, but does not exceed it. For the unstable prior, both plots end at the point the car spins out.

VIII. Conclusion

This paper presents Differential Bayesian Filtering, a novel Bayesian framework for trajectory synthesis in high-speed autonomous racing situations. Our method can be used to infer optimal trajectories from known differential properties of the optimal racing line by framing the problem as recursive Bayesian estimation on the control points of a Bèzier curve. We also present a sampling-based Monte Carlo method for DBF on Bèzier curves. We evaluate the performance of DBF using our Formula 1 simulator, and show that it results in the overall best performance, compared to other approaches and the expert driving examples, on a variety of racing metrics. Future work includes extending this approach to a multi-agent racing setup as well changing the input space of the network from pixels of images to a canonical view of autonomous driving (e.g. the current state of the ego and any other agents) as the input space.

References

[1] Matthew O'Kelly, Varundev Sukhil, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Dipshil Agarwal, Madhur Behl, Paolo Burgio, et al. F1/10: An open-source autonomous cyber-physical platform. *arXiv preprint arXiv:1901.08567*, 2019.

[2] Varundev Suresh Babu and Madhur Behl. f1tenth. dev-an open-source ros based f1/10 autonomous racing simulator. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 1614–1620. IEEE, 2020.

[3] Indy autonomous challenge. url=https://www.indyautonomous challenge.com/, journal=Indy Autonomous Challenge.

[4] Trent Weiss and Madhur Behl. Deepracing: A framework for agile autonomy. *Design, Automation and Test in Europe Conference*, 2020.

[5] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[6] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4693–4700. IEEE, 2018.

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

[8] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

[9] X. Li et al. Real-time trajectory planning for autonomous urban driving: Framework, algorithms, and verifications. *IEEE/ASME Transactions on Mechatronics*, 21(2):740–753, 2016.

[10] S. Zhang, W. Deng, Q. Zhao, H. Sun, and B. Litkouhi. Dynamic trajectory planning for vehicle autonomous driving. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4161–4166, 2013.

[11] Long Xin, Yiting Kong, Shengbo Eben Li, Jianyu Chen, Yang Guan, Masayoshi Tomizuka, and Bo Cheng. Enable faster and smoother spatio-temporal trajectory planning for autonomous vehicles in constrained dynamic environment. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 235(4):1101–1112, 2021.

[12] Thibault Buhet, Emilie Wirbel, Andrei Bursuc, and Xavier Perrotton. Plop: Probabilistic polynomial objects trajectory planning for autonomous driving. *Conference on Robot Learning (CoRL)*, 2020.

[13] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10):1497–1527, 2020.

[14] Achin Jain and Manfred Morari. Computing the racing line using bayesian optimization. *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 6192–6197, 2020.

[15] Ana I. Maqueda, Antonio Loquercio, Guillermo Gallego, Narciso N. García, and Davide Scaramuzza. Event-based vision meets deep learning on steering prediction for self-driving cars. *CoRR*, abs/1804.01310, 2018.

[16] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. Going deeper: Autonomous steering with neural memory networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 214–221, Hawaii Convention Center HI, 2017.

[17] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. *CoRR*, abs/1612.01079, 2016.

[18] F. Altché and A. de La Fortelle. An lstm network for highway trajectory prediction. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 353–359, Oct 2017.

[19] Eslam Mohammed, Mohammed Abdou, and Omar Ahmed Nasr. End-to-end deep path planning and automatic emergency braking camera cocoon-based solution. In *Machine Learning for Autonomous Driving, NeurIPS 2019 Workshop*, December 2019.

[20] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR*, abs/1812.03079, 2018.

[21] Luc Le Mero, Dewei Yi, Mehrdad Dianati, and Alexandros Mouzakitis. A survey on imitation learning techniques for end-to-end autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–20, 2022.

[22] Peide Cai, Yuxiang Sun, Yuying Chen, and Ming Liu. Vision-based trajectory planning via imitation learning for autonomous vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2736–2742, 2019.

[23] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. Super-human performance in gran turismo sport using deep reinforcement learning. *CoRR*, abs/2008.07971, 2020.

[24] Trent Weiss and Madhur Behl. Deepracing: Parameterized trajectories for autonomous racing. *IROS 2021 Workshop on Perception, Learning, and Control for Autonomous Agile Vehicles*, 2021.

[25] Trent Weiss, Varundev Suresh Babu, and Madhur Behl. Bezier curve based end-to-end trajectory synthesis for agile autonomous driving. In *NeurIPS 2020 Machine Learning for Autonomous Driving Workshop*, 2020.

[26] Ronny Hug, Wolfgang Hübner, and Michael Arens. Introducing probabilistic bézier curves for n-step sequence prediction. In *AAAI*, pages 10162–10169, 2020.

[27] Richard S Wallace, Anthony Stentz, Charles E Thorpe, Hans P Moravec, William Whittaker, and Takeo Kanade. First results in robot road-following. In *IJCAI*, pages 1089–1095. Citeseer, 1985.