Real-time Trajectory Generation via Dynamic Movement Primitives for Autonomous Racing

Catherine Weaver[1,2], Roberto Capobianco[1,3], Peter R. Wurman[1], Peter Stone[1,4], and Masayoshi Tomizuka[2]

***Abstract*— We employ sequences of high-order motion primitives for efficient online trajectory planning, enabling competitive racecar control even when the car deviates from an offline demonstration. Dynamic Movement Primitives (DMPs) utilize a target-driven non-linear differential equation combined with a set of perturbing weights to model arbitrary motion. The DMP's target-driven system ensures that online trajectories can be generated from the current state, returning to the demonstration. In racing, vehicles often operate at their handling limits, making precise control of acceleration dynamics essential for gaining an advantage in turns. We introduce the *Acceleration goal (Acc. goal)* DMP, extending the DMP's target system to accommodate accelerating targets. When sequencing DMPs to model long trajectories, our *Acc. goal* DMP explicitly models acceleration at the junctions where one DMP meets its successor in the sequence. Applicable to DMP weights learned by any method, the proposed DMP generates trajectories with less aggressive acceleration and jerk during transitions between DMPs compared to second-order DMPs. Our proposed DMP sequencing method can recover from trajectory deviations, achieve competitive lap times, and maintain stable control in autonomous vehicle racing within the high-fidelity racing game Gran Turismo Sport. Video available: `https://sites.google.com/berkeley.edu/racingdmp/home`**

I. Introduction

Autonomous racing is a complex problem of growing interest [1]. Recent advances in end-to-end reinforcement learning (RL) have achieved remarkable success in racing video games [2]; however, RL can be unstable, sensitive to tuning parameters, and require extensive environment interactions. Thus, structured trajectory planning and control methods are often preferred in real racing vehicles [1]. Online trajectory planning, in contrast to static offline planning, enhances adaptability and performance by dynamically adjusting to system changes, planning errors, or disturbances. Furthermore, control inputs can be optimized using real-time measurements of the system's state so the controller can make the best possible decision at each moment. This paper explores the topic of computationally efficient, online trajectory planning. While such online trajectory planning could benefit many robotic domains, we apply our method to racing, where precise acceleration and braking are crucial for vehicles to remain stable, navigate turns, and achieve fast lap times, all while operating at the limits of handling.

[1]Sony AI

[2]Department of Mechanical Engineering, University of California, Berkeley, CA 94720, USA. Catherine Weaver is supported by NSF GFRP Grant No. DGE 1752814. Contact: `catherine22@berkeley.edu`.

[3]Department of Computer, Control and Management Engineering, Sapienza University of Rome, 00185 Rome, Italy

[4]Department of Computer Science, The University of Texas at Austin, TX 78712, USA

To ensure safe and reliable control, planned trajectories should adhere to the system's actuation limits and dynamic constraints, but optimizing such trajectories is challenging for complex systems with real-time computation limits. Optimization and model-based methods [3]–[5] are capable of enforcing constraints and computing optimal commands, but they are computationally expensive and require accurate system models. While sampling-based methods [6] can avoid optimization routines by finding the best trajectory from a selection of potential samples, they may require additional feasibility checks or incur computational expense when many samples are needed. In racing trajectory planning, vehicle dynamics are often simplified or trajectory horizons are kept relatively short to facilitate real-time planning, but this can potentially yield sub-optimal solutions.

Motion primitives have addressed such challenges in many robotic domains by combining pre-defined basic motion patterns for low-computation trajectory generation. Specifically, Dynamic Movement Primitives (DMPs) generate movements by integrating a non-linear differential equation [7]. Unlike other geometric motion primitives [3], [8], [9], DMPs are composed of two parts: a target-driven system and a perturbing function. The target-driven system is a goal-conditioned differential equation that generates a smooth trajectory from an initial state to a desired end (goal) state. The perturbing function, or *transformation function*, provides flexibility to model state variations or system dynamics by perturbing the differential equation with a nonlinear function that is parameterized by a set of weights. While DMP-based RL has learned DMP weights that produce unseen trajectories [10], in general these methods do not account for system constraints such as dynamics and incur additional computation. Rather, we learn the weights by imitating a demonstration, since the unique two-part structure of DMPs can generate efficient trajectories with different initial or final states that retain much of the original demonstration dynamics [11].

To apply DMPs to autonomous racing, we propose a method to imitate a racing demonstration with DMPs, then plan new trajectories from real-time state measurements during control. Our approach first segments the long demonstration, and each segment is imitated by a DMP, forming a sequence of DMPs. During online model-predictive control (MPC), new DMP trajectories are generated starting from the measured state using the closest primitive in the sequence. We switch from one DMP to the next after the original time duration of the first primitive has elapsed. This adapts the so-called "target crossing" method [7], which is well-

suited for racing as it maintains the original duration of the trajectory and is very efficient to compute. The generated trajectories start from the most recent state measurement, and due to the two-part DMP structure, gradually return to the demonstration while imitating the demonstration dynamics.

Target crossing has only been performed with second-order DMPs, but these cannot sufficiently model the acceleration of a DMP sequence. For instance, in racing, precise control of lateral forces is essential for maintaining vehicle stability at the limits of handling. Vehicles can also gain an advantage in turns by employing a "slow in, fast out" racing approach, delaying braking until the last moment of entering the turn and accelerating immediately upon exiting. Therefore, trajectories that accurately control the acceleration can result in faster lap times and more stable control.

We propose the *Acceleration goal (Acc. goal)* DMP, designed to combine sequences of third-order DMPs using target crossing to generate near-optimal and near-feasible trajectories. The *Acc. goal* DMP extends the capabilities of a third-order DMP [10] by including the *acceleration* of the target state. This improvement allows for explicit modeling of acceleration at the points where a DMP in a sequence meets its successor. It is applicable to DMP weights learned by any method. Our contributions include:

1) The proposed *Acc. goal* DMP extends the target system to handle accelerating targets.
2) Our *Acc. goal* DMP plans trajectories with less aggressive acceleration and jerk when transitioning between DMPs in sequence compared to second-order DMPs.
3) We generate online DMP trajectories from a sequence of DMPs for control in Gran Turismo Sport.
4) Our method recovers from starting states where a baseline controller fails and yields less tracking error, less aggressive control, and the lowest lap-time compared to existing, non-RL, methods.

II. Acceleration Dynamic Movement Primitives

To model a trajectory as a sequence of DMPs, a DMP models the position, p_1, and its time derivative(s) of each trajectory section. In *target crossing* [7], [12], a new trajectory is generated from the DMP sequence, transitioning between DMPs based on the time index of the new trajectory. This is because, at the end of a DMP's duration, the goal-driven system has driven the trajectory close to the DMP's target state, which serves as the joining point to the next DMP [12]. Second-order DMPs with a moving target state can ensure transitions with smooth velocity [12]. We extend this notion to the target state's *acceleration* by reformulating a third-order DMP [10], allowing us to model sequences of DMPs with smooth acceleration.

A. Review of Second Order DMPs with Moving Goals

Second-order DMPs [13] model the position, p_1, and scaled velocity, p_2, of each degree of freedom as

$$\dot{p}_2 = \tau\alpha_g \left(\beta_g \left(g_m - p_1\right) + \frac{\dot{g} - \dot{p}_1}{\tau}\right) + \tau A f(z) \tag{1a}$$

$$\dot{p}_1 = \tau p_2, \tag{1b}$$

with time constant $\tau = 1/T$. A scaling coefficient A is linearly scaled with the goal and initial positions [12], and parameters α_g and β_g are set so that the system is critically damped [12]. The phase variable, z, is a first-order system, $\dot{z} = -\tau\alpha_z z$, with parameter α_z set so $z(t)$ decays from $z(0) = 1$ to $z(T) \to 0$.

The first terms of (1a) drive the system toward the goal position, g_m, which moves with constant velocity, $\dot{g}$:

$$g_m = g_m^0 - \dot{g}\frac{\ln z}{\tau\alpha_h}. \tag{2}$$

The parameter α_h and initial goal position g_m^0 are set so that the goal reaches the specified goal position at $t = T$.

The last term of (1a) represents the parameterized nonlinear system and contains the transformation function $f(z)$:

$$\begin{aligned} f(z) &= \Sigma_{i=1}^{N} \psi_i(z)\theta_i z \\ \psi_i(z) &= \frac{\exp\left(-h_i\left(z - c_i\right)^2\right)}{\sum_{j=1}^{N}\exp\left(-h_j\left(z - c_j\right)^2\right)}. \end{aligned} \tag{3}$$

The heights, h_i, and centers, c_i, of the activation functions, ψ_i, are spaced evenly in the time domain [11]. The weights, θ_i, shape the trajectory and can be learned via imitation learning or RL [7]. However, the second order formulation above does not explicitly model the demonstration's acceleration, $\ddot{p}$, which can lead to inaccurate accelerations [10].

B. Third Order DMP with Proposed Accelerating Goal

Inspired by (2), we aim to explicitly model the *acceleration* of the target. We start with a third-order DMP formulation [10], reformulated to be analogous to (1):

$$\begin{aligned} \dot{p}_3 =& \tau\alpha_p \left[\beta_p \left(\gamma_p \left(g_m - p_1\right) + \frac{\left(\dot{g} - \dot{p}_1\right)}{\tau}\right) + \frac{-\ddot{p}_1}{\tau^2}\right] \\ &+ \tau A f(z) \end{aligned} \tag{4a}$$

$$\dot{p}_2 = \tau p_3 \tag{4b}$$

$$\dot{p}_1 = \tau p_2, \tag{4c}$$

with scaled acceleration, p_3, and repeated eigenvalues set by α_p, β_p, and γ_p. We name the above formulation *Vel. goal* DMP as it employs a constant velocity goal (2). If DMPs are placed in sequence, the acceleration of joining points is assumed to be zero. While arbitrarily large θ_i can overcome modeling errors if the final state of the demonstration has non-zero acceleration, this can lead to large acceleration jumps when transitioning between DMPs in sequence.

We propose the *Acc. goal* DMP driven towards a constant acceleration, $\ddot{g}$. The evolution of the goal-driven system to the accelerating target is adapted by replacing (4a) with

$$\begin{aligned} \dot{p}_3 =& \tau\alpha_p \left[\beta_p \left(\gamma_p \left(g_m - p_1\right) + \frac{\left(\dot{g}_m - \dot{p}_1\right)}{\tau}\right) + \frac{\left(\ddot{g} - \ddot{p}_1\right)}{\tau^2}\right] \\ &+ \tau A f(z), \end{aligned} \tag{5}$$

where the moving target in (2) is replaced by

$$g_m(z) = g_m^0 + 1/2\ddot{g}\left(\frac{\ln(z)}{\tau\alpha_h}\right)^2 - \dot{g}_m^0\left(\frac{\ln(z)}{\tau\alpha_h}\right). \tag{6}$$

The initial velocity $\dot{g}_m^0$ and position g_m^0 and parameter α_h are set so that the target reaches the goal position g and velocity $\dot{g}$ at $t = T$ when moving with constant acceleration $\ddot{g}$. The *Acc. goal DMP* can be used with any DMP learning method to describe trajectories that end in non-zero acceleration. In Section IV, we join DMPs in sequence by learning θ_i from imitation [11] and setting the goal to the final state of the demonstration segment where it joins the subsequent segment. In this case, our *Acc. goal DMP* is uniquely able to model acceleration at these joining points.

III. Related Work

In this section we compare DMP methods capable of modeling *long* trajectories, focusing on those that utilize primitives with via-points or sequences of multiple motion primitives. We recognize that in racing, there are often limited demonstrations, real-time computation constraints, and significant accelerations that must be considered.

A. Motion primitives with via-points

Via-points Movement Primitives (VMPs) enable primitives to have multiple via-points by representing the target state as a function [14]. However, they can introduce large accelerations that may be undesirable for robot [7] or vehicle motion. Via-points are also possible with Probabilistic Movement Primitives [15] and Kernelized Movement Primitives (KMPs) [16]; however, sufficient demonstrations are required to model a probability distribution, and a reasonable covariance matrix on which to condition new via points must be calculated. Such probabilistic modeling is difficult in racing, where there is limited access to demonstrations.

B. Sequences of DMPs

Sequences of DMPs join multiple motion primitives in series to model a full trajectory [12], [17], [18]. The *basis functions overlay* sequencing method [7], [18], [19] combines DMPs using overlapping basis functions and a sigmoidal phase variable. However, computation time increases with the number of DMPs, and generated trajectories last longer than demonstrations due to the alternate phase variable [19], which is unfavorable in domains where time to complete the trajectory is important (such as racing). Conversely, the *target crossing* sequencing method [7], [12], switches between DMPs when the time reaches the duration of the original demonstration; this maintains the original trajectory duration and does not increase computation with more DMPs in sequence. A constant velocity goal state [12] joins DMPs at points with non-zero velocity, but the second-order system may lead to undesirable behavior in domains requiring precise acceleration control [10].

C. Third Order DMPs

Higher order DMPs can generate more reasonable trajectories than the commonly-used second order system [7]. Nemec and Ube [20] sequenced the third order "acceleration" DMP [21], which placed a filter on the goal system of a second order DMP [11]. The formulation produces smooth acceleration when switching between DMPs. However, the attractor point is restricted by the filter system, so the formulation does not model sequences connected at points with arbitrary velocities and accelerations. Another third-order DMP [10] raised the order of the goal-driven system, perturbing the jerk instead of the original acceleration [11]. This formulation converges to a constant velocity goal [12] and has not been used for sequences of DMPs. Our work extends the goal to move with constant acceleration, enabling target crossing to sequence DMPs joining at states with non-zero acceleration.

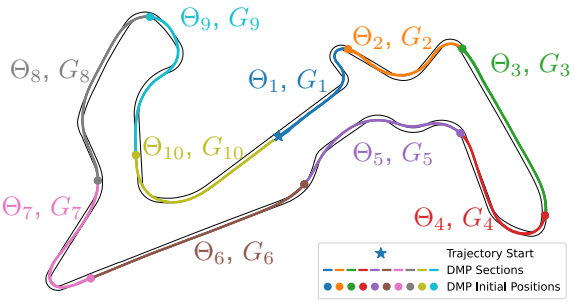


Fig. 1. DMP sections of duration T_S for a demonstration trajectory on Lago Maggiore racetrack; DMP weights, Θ_j, and goal states, G_j, for the jth segment are learned via imitation

IV. Sequence of DMPs Trajectory Generation

Our method first learns a sequence of DMPs from a demonstration, P_d, then generates new trajectories from that sequence. Our method assumes access to a demonstration, such as the trajectory in Fig. 1. We compare the trajectories generated by our *Acc. goal* DMP (5) to two baselines: the *2nd. order* DMP (1) and third order *Vel. goal* DMP (4).

A. Imitation Learning of the Sequence of DMPs

The demonstration is first imitated with a sequence of DMPs. The demonstration is divided into segments of equal duration, T_S. The demonstration time is normalized with respect to each segment, so that for the m-th waypoint, the time $t_m \in [0, T_S)$ denotes the time since the start of the segment. Hence each segment starts at $t_m = 0$ and ends at $t_m = T_S$. As shown in Fig. 1, each segment is modeled by the DMP weights, $\Theta_j = ([\theta_0 \dots \theta_i \dots \theta_N]^T)_j$, and the DMP target state $G_j = \{g, \dot{g}, \ddot{g}\}$, where j is index of the segment, and N denotes the number of weights per segment per degree of freedom. The target state, G_j, is set to the final state of the demonstration segment at $t_m = T_S$. For each baseline, the weights, Θ, are learned via local weighted regression [11]. In the racing example, we divide the demonstration into 10 segments, so that the segment duration, $T_S = 11.25$s, is long enough to prevent rapid switching between segments but short enough to accurately model the demonstration.

B. Sequences of DMPs Trajectory Generation Method

Algorithm 1 uses the sequence of DMPs to generate ***new*** trajectories, P_G, of duration T_G from an initial state, p_0. We determine the index, m, of the closest waypoint in the

Algorithm 1 Sequence of DMPs Trajectory Generation

1: **Input**: initial state $P_i = \{p_i, \dot{p}_i, \ddot{p}_i\}$, demonstration $P_d = \{P_{d,0}, \ldots, P_{d,M}\}$, and time step Δt
2: **Output** $P_G = \{p_G, \dot{p}_G, \ddot{p}_G\}$ Generated trajectory
3: $P_G^{t=0} \leftarrow P_i$
4: $m = \text{argmin}_m \|p_{d,m} - p_i\|_2$
5: $j \leftarrow j(m)$, $G \leftarrow G_j$, $\Theta \leftarrow \Theta_j$
6: $t_m \leftarrow t(m) \in [0, T_S)$ ▷ Time of demonstration at m
7: **for** $t_i = 0, \ldots, T_G$ **do**
8: Calculate $\dot{P}_G^{t_i}(G, \Theta, z_i)$ at $z_i = z(t_m + t_i)$ using DMP equations (i.e. Eqs. 1, 4, or 5)
9: Integrate trajectory $P_G^{t_{i+1}} = P_G^{t_i} + \dot{P}_G^{t_i} \Delta t$
10: **if** $t_i + t_m \geq T_G$ **then** ▷ Start next segment
11: $G \leftarrow G_{j(m)+1}$
12: $\Theta \leftarrow \Theta_{j[i]+1}$
13: $t_m \leftarrow 0$
14: **end if**
15: **end for**

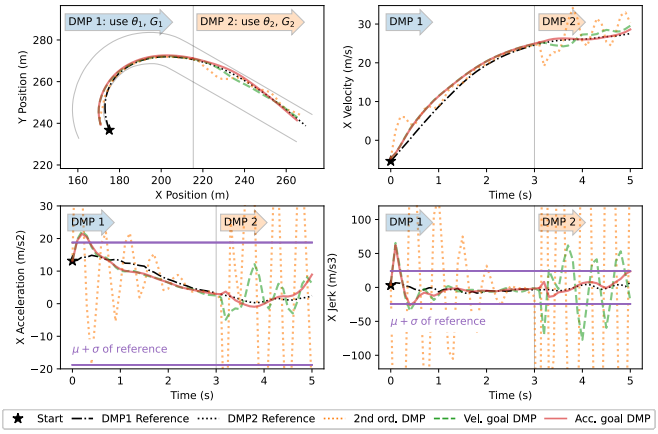
demonstration to the initial state (Line 4). This waypoint corresponds to the j-th segment and time t_m. The trajectory is generated by integrating the DMP equations (Eqns. 1, 4, and 5) with respect to time starting at t_m (Line 8). Sometimes the initial state is close enough to the end of the j-segment that during integration the generated trajectory reaches the end (in time) of this segment. When this occurs (Line 10), we are inspired by the target crossing method to reset the time (Line 13) and generate the remainder of the trajectory with the subsequent DMP's information (G_{j+1} and Θ_{j+1}). For our racing experiments, we set $T_G = 5$s, which provides ample information about the future trajectory and guarantees the DMP is switched (Line 10) at most once, since $T_G < T_S$.

C. Results: Racing Trajectory Imitation

We first analyze how well each DMP method imitates a racing demonstration of 6455 waypoints (Fig. 1. It is generated with an expert RL policy from an agent in Wurman et al. [2] on the Lago Maggiore GP racetrack and vehicle

TABLE I
SEQUENCE OF DMPS' IMITATION ERROR TO DEMONSTRATION

	Mean (Std. Dev.) of Tracking Error			
Primitive	**Position**	**Velocity**	**Acceleration**	**Jerk**
$N = 23$ weights per DMP segment				
2nd ord. DMP	1.05 (1.5)	3.73 (4.7)	35.05 (42.4)	400.24 (502.3)
Vel. goal DMP	1.23 (2.1)	1.41 (1.9)	4.67 (5.8)	43.6 (50.9)
Acc. goal DMP	1.36 (2.0)	1.41 (1.7)	4.13 (5.8)	37.01 (50.6)
$N = 184$ weights per DMP segment				
2nd ord. DMP	0.59 (1.2)	1.25 (2.4)	5.43 (8.4)	78.95 (104.0)
Vel. goal DMP	1.16 (2.0)	1.09 (1.7)	1.82 (2.6)	13.25 (13.3)
Acc. goal DMP	1.16 (2.0)	1.08 (1.7)	1.8 (2.6)	13.05 (13.0)



(a) New trajectory generation from state OFF demonstration

Fig. 2. Trajectory generation for duration of $T_G = 5$ s starting 3 seconds before switching DMP segments with $N = 23$ weights per segment; $\mu + \sigma$ indicates the mean+standard deviation of the magnitude of the demonstration acceleration and jerk.

equivalent to the Federation Internationale de l'Automobile (FIA) GT3 class. The sequence of DMPs is determined in Section IV-A; then trajectories are generated via Algorithm 1 from every point on the demonstration. The demonstration tracking error is in Table I. We compare different number of DMP weights per segment, and find that $N = 23$ models the demonstration with low enough position, velocity and acceleration error for control. Our Acc. goal DMP reduces the tracking error of velocity, acceleration, and jerk with slightly worse position error than the Vel. goal DMP.

D. Analysis: Trajectory Generation off of Demonstration

A notable advantage of the sequence of DMPs is the ability to generate trajectories from unseen states that can ***return*** to the demonstration. Fig. 2 shows trajectories generated by each DMP type from an initial position off of the demonstration (the trajectories' initial velocity and acceleration are set to those of the closest point on the demonstration.) The initial position is 3 seconds before the end of the current segment. Per Algorithm 1, the dynamics of the first DMP (DMP1) are integrated until 3 seconds has elapsed; then the weights/goal state of the subsequent DMP (DMP2) are used to integrate the remainder of the trajectory. The DMPs' goal-driven system drives the trajectory towards the goal state, so the generated trajectory returns to the demonstration. Meanwhile, the DMPs' transformation function mimics the original dynamics of the demonstration.

The higher order DMPs (*Vel. goal* and *Acc. goal*) reduce deviation from the demonstration by explicitly modeling acceleration, in comparison to the *2nd ord. DMP* previously used for target crossing [12]. However, while the *Vel. goal* DMP can compensate with arbitrarily large weights at the end of DMP1, transitioning from DMP1 to DMP2 introduces a large acceleration and jerk due to the non-zero crossing acceleration. Our *Acc. goal DMP* accounts for this non-zero crossing acceleration, resulting in a smoother trajectory at the crossing point. In fact, only the *Acc. goal DMP*'s

acceleration and jerk are comparable in magnitude to those in the demonstration (mean plus standard deviation of these values for the demonstration in purple in Fig. 2).

V. Real-Time Trajectory Planning and Control

We employ an MPC [22] that observes the most recent state, x, and minimizes tracking error over a prediction horizon, T_C. However, real-time computation limits T_C, make it difficult or impossible to find a feasible solution from distant states if the MPC tracks the offline demonstration. To address this, our MPC tracks real-time trajectories that start from the last observed state and gradually return to the demonstration over a longer horizon T_G. These online MPC reference trajectories are generated using a sequence of DMPs (Section IV), and these computationally efficient DMP trajectories permit T_G to be much longer than T_C. In this section, we describe the rules for generating DMP trajectories during control and summarize the MPC.

A. Controller Logic for DMP Trajectory Generation

To ensure stable control and reduce computation, new MPC references (i.e. DMP trajectories) are not generated continuously from every observed state. Rather, a new trajectory is only generated when

1) the state is near the end of the MPC reference; i.e. it has been $T_G - T_C - t_b$ since the last trajectory was generated, or
2) the state has deviated a large distance D from the MPC reference.

Condition (1) ensures that the MPC reference (of length T_G) is always longer than the MPC control horizon, T_C, including a small buffer, $t_b = 1$s. Condition (2) provides a new trajectory whenever the observed state deviates significantly from the current MPC reference. A DMP trajectory is always generated from the initial state.

B. Nonlinear Model Predictive Control

The controller minimizes tracking error to the DMP trajectories. A limitation of the DMP trajectories is the lack of vehicle heading information – only global position and its derivatives are modeled. A kinematically-constrained DMP has been proposed [23], but a kinematic model is overly restrictive for racing vehicles for which a dynamic model with nonlinear tire dynamics is more accurate [24]. We compensate for this limitation by using a nonlinear MPC that constrains the states and controls to obey a nonlinear dynamic model, $x_{t+1} = f(x_t, u_t)$ (Appendix).

The MPC minimizes cost function $J(\mathbf{x}, \mathbf{u})$, where $\mathbf{x}$ and $\mathbf{u}$ denote the state and control sequences over horizon T_C:

$$J(\mathbf{x},\mathbf{u}) = \sum_{t=0}^{T_C-1} \mathbf{w}_D^T \Big[\|\delta p\|_t^2, \quad \|\delta \dot{p}\|_t^2, \quad \|\delta \ddot{p}\|_t^2, \quad \dots \quad v_{y,t}^2, \quad u_{\delta,t}^2, \quad u_{\text{th/br},t}^2 \Big]^T, \tag{7}$$

where $\mathbf{w}$ denotes the cost function weights, $p, \dot{p}$, and $\ddot{p}$ denote 2-D position and its time derivatives, and u_δ, and $u_{\text{th/br}}$ denote steering angle and throttle/brake input respectively. The $\delta(\cdot)$ function indicates the difference between the state and the reference, e.g. $\delta p = p - p_{ref}$. We regularize the lateral velocity v_y calculated by the car model (Appendix) to stabilize the system by preventing large lateral motion.

As a baseline, we use the offline demonstration as the MPC reference. Since the demonstration contains vehicle heading and control information, a fair cost function is

$$J(\mathbf{x},\mathbf{u}) = \sum_{t=0}^{T_C-1} \mathbf{w}_C^T \big[\|\delta p\|_t^2, \quad \delta v_{x,t},^2 \quad \dots \quad v_{y,t}^2, \quad \delta u_{\delta,t}^2, \quad u_{\text{th/br},t}^2 \big]^T, \tag{8}$$

where v_x is longitudinal velocity.

The minimization of J is solved with an iterative linear quadratic regulator (iLQR) [22], [25]. Jacobians A_t and B_t are estimated with `tensorflow.GradientTape` and the LQR solution δu_t^* is found via the Riccati recursion [25]. MPC's receding horizon re-calculates the optimal control at each state and only applies the first step of the solution, u_0^*.

VI. Control Experiments

We design two experiments to test the efficacy of DMP trajectories as references for an MPC. The experiments are conducted in the PlayStation 4 (PS4) game Gran Turismo Sport (GTS) (https://www.gran-turismo.com/us/), developed by Polyphony Digital, Inc. with racetrack, vehicle, and demonstration from Section IV-C. GTS takes two continuous inputs: throttle/braking and steering. The vehicle positions, velocities, accelerations, and pose are observed. We compare four types of MPC references: offline demonstration using (8) (denoted "Fixed Ref.") and online DMP trajectories using (7) and the three DMP methods (2nd. order, Vel. goal, and Acc. goal DMPs). To tune MPC weights for each method, an evolutionary algorithm [26] maximizes the total course progress over 400s in GTS, i.e. $\max_{\mathbf{w}_D \text{ or } \mathbf{w}_C} \sum_{t=0}^{400} s(t)$.

Model-free RL has recently achieved super-human performance in GTS [2]. While MPC approaches do not currently reach that level of performance, GTS offers a highly realistic environment for evaluating DMP trajectories. We discovered that Algorithm 1 is a reliable method for generating trajectories, even from states off of the demonstration, and our *Acc. goal* DMP outperforms the other DMP baselines. More details can be found in the accompanying video.

A. Experiment 1: Recovering from Initial Deviation

The first experiment tests our trajectory generation method and assesses if each DMP type can generate MPC references that enable the car to recover from deviations from the demonstration. The racetrack is divided into points spaced evenly in time along the demonstration; at each point, the initial position of the vehicle is set to the outermost left, outermost right, or center of the track (Figure 3). Aside from position, the other initial states (i.e. heading and velocities) are set to match the states of the point closest to the vehicle's initial position on the demonstration. The vehicle is controlled using the controller in Section V.

We must first define when a car *recovers* from an initial deviation from the demonstration. Since the demonstration is

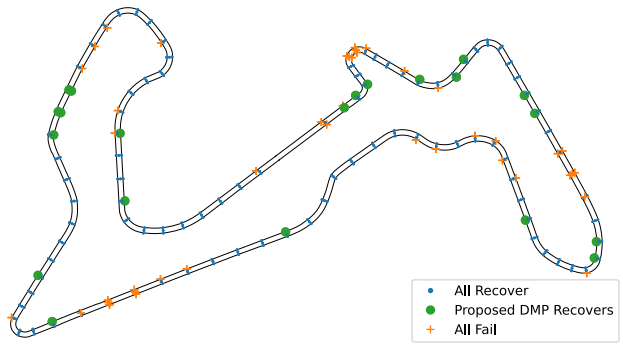


Fig. 3. Initial points of recovery experiment in GTS. Three trials are conducted from each initial point. Blue indicates initial points where all methods recover in all three trials. Green indicates initial points where the MPC recovers using Acc. goal DMP, but Fixed Ref. MPC with offline demonstration does not. Orange indicates that none of the methods recover.

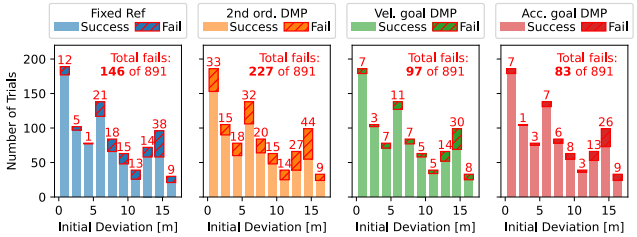
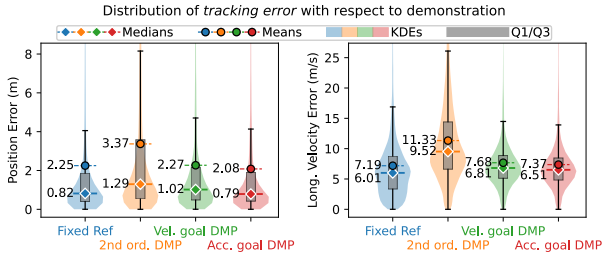


Fig. 4. Successful recoveries and failed recoveries of the MPC tracking each reference. Three control trials occur from each initial point (Figure 3); recovery occurs when $\|p_1 - p_{\text{fix,ref}}\| \leq \epsilon_{\text{pos}}$ for t_ϵ. Trial fails if vehicle fails to recover within $T_{\text{trial}} = 14$ s.

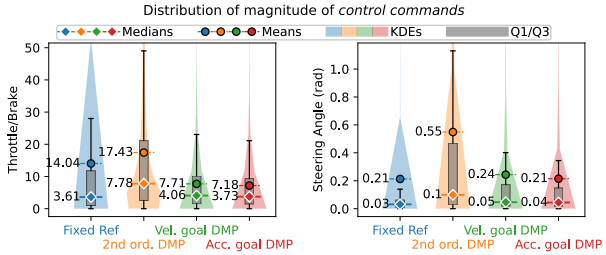
a nonlinear, time-varying function, we use the *settling time*, $t_{\text{set}} = \mathcal{T} - t_0$, to define this. It represents the time required for a system that is initially in an equilibrium state and is disturbed by an input over the interval $0 \leq t \leq t_0$ to return to within a neighborhood $\epsilon > 0$ of the equilibrium state at time $\mathcal{T}$ [27]. Inspired by this definition, we define a successful *recovery* as the controlled vehicle remaining within $\epsilon_{\text{pos}} = 1.5$ m of the demonstration for at least $t_\epsilon = 4$ seconds; i.e. if the position error remains under $\|p_1 - p_{1,\text{fix,ref}}\| \leq \epsilon_{\text{pos}}$ for the full duration $t_{\text{set}} \leq t \leq t_{\text{set}} + t_\epsilon$, we say the vehicle *recovered* at time t_{set}. Since it is also important that the vehicle recovers relatively quickly from deviations, we stipulate that a successful recovery only occurs when $t_{\text{set}} \leq t_{\text{set,max}} = 10$ s, so each trial is conducted for $T_{\text{trial}} = t_{\text{set,max}} + t_\epsilon = 14$ s. The values $\epsilon_{\text{pos}}, t_\epsilon$, and $t_{\text{set,max}}$ are generously chosen based on preliminary trials to allow each method plenty of time to return to the demonstration. Three trials for each reference type are conducted from each starting point in Fig. 3.

B. Experiment 1: Recovery Results

The effectiveness of each trajectory method, across all trials originating from each point in Fig. 3, is presented in Fig. 4. Intuitively, as the initial deviation from the demonstration increases, the MPC tracking the demonstration (Fixed Ref. MPC) is less likely to recover (16.4% failure rate). The 2nd ord. DMP is even less likely to recover, but the third order methods notable reduce the failure rate with



(a) Position and longitudinal velocity errors w.r.t offline demonstration – terms 1 and 2 in (8), respectively



(b) Magnitude of control commands.

Fig. 5. Distribution of control metrics during Experiment 1. Q1/Q3 represent 25th/75th percentile (box plot); KDE is kernel density estimation (violin plot)[3]

statistical significance,[1] showing that higher-order DMPs are essential for generating practical and effective trajectories in the context of racing. While the Acc. goal DMP tends to fail less than the Vel. goal DMP (9.3% vs. 10.9%), the variation in their success rates is not statistically significant.[2]

C. Experiment 1: Analysis

We employ closed loop control metrics as indicators of how well-suited the generated trajectories are for control. For every observed state during Experiment 1, we calculate the tracking error with respect to the demonstration and the magnitude of the control commands. For tracking error, we focus on position and longitudinal velocity error (terms 1 and 2 in Eq. 8), as these are critical for racing performance.

We provide the distribution of the control metrics in Fig. 5 in the form of a violin plot, which is similar to a box plot with distribution estimation using kernel density estimation (KDE).[3] The distributions of all methods are highly skewed by instances with unreasonable commands due to large deviations from which recovery may not be possible (e.g. starting points with orange crosses in Fig. 3). If a method fails immediately (crashes into a wall or runs very far off the track), the 14-second trial is stopped early. So even thought the Fixed Ref. MPC has marginally smaller tracking error, instances where the MPC fails immediately are not taken into account. The Vel. and Acc. goal DMP have much better recovery performance than the Fixed Ref. MPC at starting

[1]Measured success rate of Vel. goal DMP compared to Fixed Ref. MPC: $\chi^2(2, N = 1782) = 11.4, p < .001$. Acc. goal DMP vs. Fixed Ref. MPC: $\chi^2(2, N = 1782) = 19.9, p < .001$.

[2]Acc. vs. Vel. goal DMP success rate: $\chi^2(2, N = 1782) = 1.2, p = .27$.

[3]Refer to `matplotlib.pyplot.violinplot`.

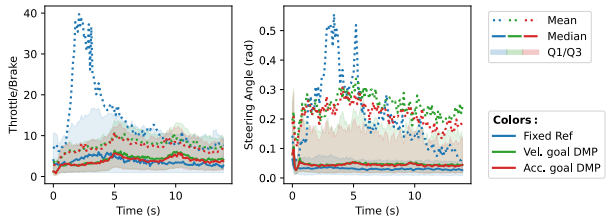


Fig. 6. Distribution of the magnitude of control commands over the horizon of each trial in Experiment 1. Q1/Q3 represent 25th/75th percentile.

TABLE II

GT LAP TIMES FOR EACH TRAJECTORY METHOD

Trajectory Method with MPC	Min Lap-time	Mean (Std.)	# of Laps
MPC + 2nd order DMP	155.733	158.4 (2.18)	4
MPC + Vel. goal DMP	133.95	136.87 (1.34)	28
MPC + Acc. goal DMP	132.966	134.75 (0.85)	29
MPC + Fixed Ref.	130.283	131.87 (0.92)	21
Demonstration Lap-time		*114.5*	

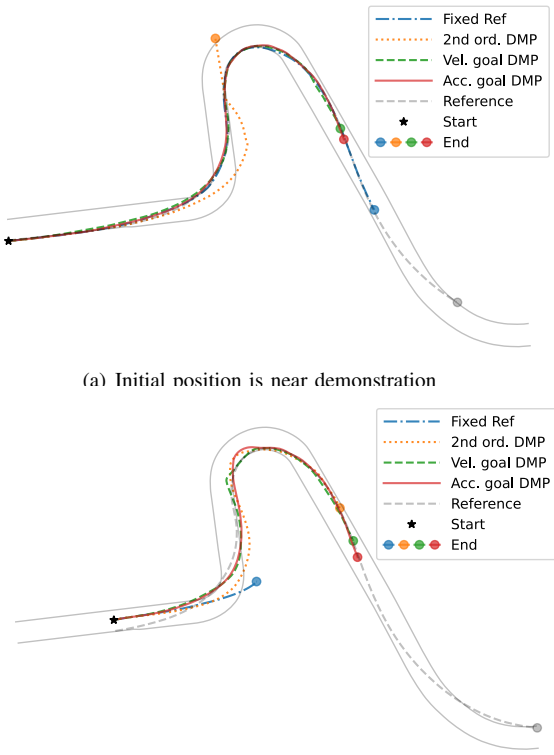
positions away from the demonstration (Fig. 4). At these difficult starting positions, the recovered paths of the Vel. and Acc. goal DMP may deviate more from the demonstration and contribute to the slightly larger tracking error in Fig. 5. Even so, the DMP trajectories produce more reasonable control commands compared to the Fixed Ref. MPC, and in particular, the Acc. goal DMP exhibits the smallest tracking error and smallest control commands. Fig. 6 provides the distribution of control commands over the 14-second horizon that the car has to recover. The Fixed Ref. MPC generates more drastic commands, especially at the beginning of the control horizon, compared to both Vel. and Acc. goal DMPs. *If* the Fixed Ref. MPC recovers, it eventually reduces to more reasonable commands, but the Vel. and Acc. goal DMPs do not see this initial spike to unreasonable control values.

D. Experiment 1: Individual Trajectory Analysis

We analyze two initial positions near a corner encountered during Experiment 1; from one, the car starts near the demonstration, and for the other, it starts away from the demonstration. Each method's best trial (most course progress) is shown in Fig. 7. When the vehicle starts near the demonstration (Fig. 7a), the Fixed Ref. MPC has the most course progress and outperforms MPC+DMP methods due to the explicit availability of vehicle-oriented states (Section V-B). However, when the initial position is away from the demonstration (Fig. 7b), the Fixed Ref. MPC does not find a reasonable solution within control horizon T_C and fails almost immediately. In contrast, the online DMP trajectories slowly return to the demonstration over a longer horizon T_G, allowing the MPC to track the more reasonable trajectory and recover from the deviation to pass through the corner. Notably, our Acc. goal DMP allows the most progress through the corner by best modeling acceleration.

E. Experiment 2: Time-trial Track Race

A full-track experiment was also performed in GT, with the lap-time results in Table II. The state of the vehicle is



(a) Initial position is near demonstration

(b) Initial position is away from demonstration

Fig. 7. Best (most course progress) trial results for each method from slightly different starting positions near same corner.

initialized to the first state of the demonstration at the start of the race, and the vehicle is given 400 seconds to complete as many laps as possible over 11 trials. If the vehicle spins backwards or goes significantly off-course, the trial is stopped. The MPC with 2nd ord. DMP trajectories exhibits the worst performance, as they cannot adequately model acceleration. This leads to higher lap-times and frequent spinning out before completing a lap. The Fixed Ref. MPC achieves the best lap-time and the lowest average lap-time since it can exploit the knowledge of heading and control information (8), but it is more prone to losing stability, resulting in a lower number of completed laps (21 laps). The higher order DMPs complete the most laps (28 and 29 laps) and yield lap-times a few seconds slower than the Fixed Ref. MPC. Our Acc. goal DMP achieves the lowest lap-times of the DMP methods by generating the most suitable trajectories for racing (Section IV-D)[4].

VII. CONCLUSION

We introduced a DMP with target state acceleration, the Acc. goal DMP, and combined motion primitives in sequence to imitate a long, complex racecar demonstration. Our DMP trajectory generation method produces trajectories suitable for control that aid an MPC in recovering from deviations

[4]The difference in mean lap-times between the Acc. goal DMP ($M = 134.75, SD = .85$) and Vel. goal DMP ($M = 136.87, SD = 1.34$) was statistically significant: $t(55) = 7.16, p < .001$. The 95% confidence interval for the difference in means is $\mu_{Acc.goal} - \mu_{Vel.goal} \in [-2.7, -1.5]$.

from the demonstration. During real-time control in the complex racing game Gran Turismo Sport, we assess our method with two experiments designed to test a vehicle's ability to recover from deviations and achieve the best lap-time. An MPC used our DMP trajectories as a reference to recover from deviations from the demonstration, outperforming an MPC using only the offline demonstration. Thus, the DMPs allow the MPC to be more robust to leaving the demonstration in states during control.

We note that third order DMPs (Vel. and Acc. goal) were critical for performance, even though second order DMPs are most commonly used. Our proposed Acc. goal DMP explicitly models non-zero acceleration at joining points in a sequence of DMPs; thus it performs the best of the DMP methods in terms of recovery, tracking error, and lap-time. The proposed DMP formulation and trajectory generation framework allows DMP sequences with non-zero accelerations at joining points, and thus can imitate complex racing demonstration. However, DMPs have also been used widely with RL, and future works will focus on extending the Acc. goal DMP to learn trajectories that obey additional constraints, improve performance in closed loop control, or race against opponents in a multi-agent setting.

APPENDIX: NONLINEAR DYNAMIC CAR MODEL

The model $x_{t+1} = f(x_t, u_t)$ estimates the state derivative $\dot{x}_t$ at each time step and updates $x_{t+1} = \dot{x}_t \Delta t$. The state, $x = \begin{bmatrix} X & Y & v_x & v_y & \psi & \omega \end{bmatrix}^T$, contains the global coordinates X and Y, longitudinal and lateral velocity v_x and v_y, and yaw angle and rate ψ and ω. By the definition $\omega = \dot{\psi}$, $\dot{X} = v_x \cos(\psi) - v_y \sin(\psi)$, and $\dot{Y} = v_x \sin(\psi) + v_y \cos(\psi)$. Equation (1) from [24] is used to calculate $\dot{v}_x$, $\dot{v}_y$, and $\dot{\omega}$. The steering angle, u_δ, and the longitudinal acceleration, u_a, correspond to δ and α in [24], respectively. Lateral tire forces on the front F_{yf} and rear F_{yr} tires are found with a hyperbolic tangent function [28] as a function of front and rear slip angles α_f and α_r [29] using $F_{yf} = a_{f1} \mu \tanh(a_{f2} \alpha_f)$ and $F_{yr} = a_{r1} \mu \tanh(a_{r2}, \alpha_r)$. The coefficient of friction is μ, and a_{f1}, a_{f2}, a_{r1} are coefficients. The control input to the model is $u = \begin{bmatrix} u_\delta & u_{\text{th/br}} \end{bmatrix}^T$, where $u_{\text{th/br}} \in [-1, 1]$ is the scaled throttle opening ($u_{\text{th/br}} \in [0, 1]$) or brake ($u_{\text{th/br}} \in [-1, 0)$). We approximate $u_a = k_u u_{\text{th/br}} + k_V v_x$, with coefficients k_u and k_V. All model parameters are found by minimizing modeling error to the demonstration.

REFERENCES

[1] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle, M. Behl, V. Krovi, and R. Mangharam, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open J. Intell. Trans. Syst.*, vol. 3, pp. 458–488, 2022.

[2] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs, *et al.*, "Outracing champion gran turismo drivers with deep reinforcement learning," *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[3] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, L. Hermansdorfer, T. Stahl, T. Herrmann, and M. Lienkamp, "A software architecture for the dynamic path planning of an autonomous racecar at the limits of handling," in *Proc. IEEE Int. Conf. Connected Veh. Expo (ICCVE)*, 2019, pp. 1–8.

[4] H. Guo, C. Shen, H. Zhang, H. Chen, and R. Jia, "Simultaneous trajectory planning and tracking using an mpc method for cyber-physical systems: A case study of obstacle avoidance for an intelligent vehicle," *IEEE Trans. Ind. Informatics*, vol. 14, pp. 4273–4283, 2018.

[5] J. K. Subosits and J. C. Gerdes, "From the racetrack to the road: Real-time trajectory replanning for autonomous driving," *IEEE Trans. Intell. Veh.*, vol. 4, no. 2, pp. 309–320, 2019.

[6] Z. Ajanovic, E. Regolin, G. Stettinger, M. Horn, and A. Ferrara, "Search-based motion planning for performance autonomous driving," in *Proc. Int. Symp. Dyn. Veh. Roads Tracks (IAVSD)*. Springer, 2019, pp. 1144–1154.

[7] M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic movement primitives in robotics: A tutorial survey," *arXiv preprint arXiv:2102.03861*, 2021.

[8] C. You and P. Tsiotras, "High-speed cornering for autonomous off-road rally racing," *IEEE Trans. Contr. Syst. Technol.*, vol. 29, no. 2, pp. 485–501, 2019.

[9] C. L. Bottasso, D. Leonello, and B. Savini, "Path planning for autonomous vehicles by trajectory smoothing using motion primitives," *IEEE Trans. Contr. Syst. Technol.*, vol. 16, no. 6, pp. 1152–1168, 2008.

[10] X. Guo, "Trajectory generation using reinforcement learning for autonomous helicopter with adaptive dynamic movement primitive," *Proc. Institution Mech. Eng., Part I: J. Syst. Control Eng.*, vol. 231, no. 6, pp. 495–509, 2017.

[11] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural Computation*, vol. 25, no. 2, pp. 328–373, 2013.

[12] J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters, "Movement templates for learning of hitting and batting," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2010, pp. 853–858.

[13] J. Kober and J. Peters, "Policy search for motor primitives in robotics," *Machine learning*, vol. 84, no. 1, pp. 171–203, 2011.

[14] Y. Zhou, J. Gao, and T. Asfour, "Learning via-point movement primitives with inter-and extrapolation capabilities," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*. IEEE, 2019, pp. 4301–4308.

[15] A. Paraschos, C. Daniel, J. R. Peters, and G. Neumann, "Probabilistic movement primitives," *Adv. Neural Inf. Process. Syst.*, vol. 26, 2013.

[16] Y. Huang, L. Rozo, J. Silvério, and D. G. Caldwell, "Kernelized movement primitives," *Int. J. Robot. Research*, vol. 38, no. 7, pp. 833–852, 2019.

[17] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, "Learning and generalization of motor skills by learning from demonstration," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2009, pp. 763–768.

[18] T. Kulvicius, K. Ning, M. Tamosiunaite, and F. Wörgötter, "Modified dynamic movement primitives for joining movement sequences," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011, pp. 2275–2280.

[19] M. Saveriano, F. Franzel, and D. Lee, "Merging position and orientation motion primitives," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019, pp. 7041–7047.

[20] B. Nemec and A. Ude, "Action sequencing using dynamic movement primitives," *Robotica*, vol. 30, no. 5, pp. 837–846, 2012.

[21] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, "Control, planning, learning, and imitation with dynamic movement primitives," in *Workshop Bilateral Paradigms Humans Humanoids: Int. Conf. Intell. Robots Syst. (IROS)*, 2003, pp. 1–21.

[22] J. Chen, W. Zhan, and M. Tomizuka, "Constrained iterative lqr for on-road autonomous driving motion planning," in *Proc. IEEE Int. Conf. Intell. Transport. Syst.(ITSC)*, 2017, pp. 1–7.

[23] R. S. Sharma, S. Shukla, H. Karki, A. Shukla, L. Behera, and K. Venkatesh, "Dmp based trajectory tracking for a nonholonomic mobile robot with automatic goal adaptation and obstacle avoidance," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2019, pp. 8613–8619.

[24] F. K. Pour, D. Theilliol, V. Puig, and G. Cembrano, "Health-aware control design based on remaining useful life estimation for autonomous racing vehicle," *ISA Trans.*, vol. 113, pp. 196–209, 2021.

[25] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems." vol. 1, 01 2004, pp. 222–229.

[26] N. Hansen, "The cma evolution strategy: A tutorial," *arXiv preprint arXiv:1604.00772*, 2016.

[27] D. E. Frank and S. A. Musa, "Stabilities and settling times of nonlinear and time-varying feedback systems," *J. Franklin Institute*, vol. 294, no. 3, pp. 155–166, 1972.

[28] S. Stepanyuk, R. Bruns, and K. Krivenkov, "Empirical lateral-force-model for forklift tires," *Logistics Research*, vol. 10, pp. 1–12, 2017.

[29] H. Pacejka, *Tire and vehicle dynamics*. Elsevier, 2005.