Multi-Vehicle Autonomous Racing with Learning MPC and Trajectory Forecasting

Rohan Sinha
dept. of Aeronautics and Astronautics
Stanford University
Stanford, USA
rhnsinha@stanford.edu

***Abstract*—We propose adapting the Learning Model Predictive Control (LMPC) [1] algorithm to a multi-vehicle autonomous racing problem. Instead of taking a game-theoretic approach to reason intelligently about strategic interactions between agents, we hypothesise that the influence of an autonomous agents' decision making on its opponents can be learned safely online through competitive interactions provided iterative updates to the agent's policy are small. We segment the control tasks into separate elements in charge of maintaining safety, long-term strategic behavior, and avoiding opponents. We showcase our method on a simulated example of a race between two autonomous cars and conclude that trajectory forecasting holds promise for establishing intelligent competitive behavior between agents.**

***Index Terms*—predictive control, autonomous racing, behavior prediction, multi-agent systems**

I. Introduction and Motivation

To reliably deploy the next generation of autonomous robotic systems in unstructured open-world environments, they must safely interact with other agents such as humans and other robots. In many of these settings, such as when an autonomous vehicle must merge into traffic on a highway on-ramp, the autonomous agent cannot communicate with other agents to form a collaborative strategy. More importantly, automated systems will often have to compete with other agents to force their objective. Therefore, this requires the development of strategies that intelligently reason about the effect that the decisions made by an automated robot have on the actions of the agents in its environment. In this work, we propose to study a competitive interaction between an automated system and other agents.

In particular, we focus on competitive racing with self-driving cars. Autonomous racing has received considerable attention in the robotics and control community in recent years, as it is a multi-faceted problem that highlights many of the challenges faced by modern autonomous systems [2]–[7]. By developing algorithms for problems that push modern autonomy systems to their computational and physical limits, we are likely to gain insights and invent algorithms that will improve the safety and performance of autonomous systems applied in the real world. Autonomous racing is particularly difficult for four main reasons.

Firstly, modelling the dynamics of a vehicle at its tractive and handling limits is a challenging problem. The interactions between the tires of a vehicle and the road surface are complex and nonlinear [2], [3]. Due to the high speeds involved, automated racing pushes deep into these nonlinear regimes. Track conditions may vary from corner to corner, significantly affecting these dynamics. Therefore, a mis-calibrated model could cause the vehicle to push past the limits of static friction with catastrophic results. Uncertainty in the models must be taken into account and resolved through online learning to deal without ever exceeding the handling limits.

Secondly, it is not obvious what the ideal racing line is, even in a single-agent racing setting. Ideally, an agent with perfect knowledge of the track conditions and its dynamics could attempt to solve a tough nonconvex optimization problem to determine the minimum-time trajectory to cross the finish line. Even though solving such a problem exactly is intractable, with modern computational resources such a trajectory can be planned as a reference up-front using heuristics such as contouring costs [3], [6] or to a local minimum using some sequential programming technique. However, if the dynamics turn out to be slightly different in real-life, it is not realistic to recompute the reference online. Therefore, racing algorithms should be able to learn a locally optimal racing line through experience, just like Formula 1 drivers practice on a track before a race.

Thirdly, automated vehicles must both avoid collisions with opponents and reason strategically about their opponents' behavior. For example, existing work generally assumes the other agents act rationally with respect to some cost function and adopt a game theoretic approach to reason about the impact of an autonomous agent's behavior on the other actors in the system [5], [8]. Such approaches seem ideal for racing problems, as a naive approach that makes predictions on another agent's behavior and plans around those predictions won't be able to exhibit the blocking and cutting-off behavior necessary to win in racing. However, solving for policies that form a Nash equilibrium is often intractable and tends to rely on iterative approximation schemes that can be a burden for real-time control. Moreover, these approaches often assume the cost function of the opponents is known, even though it is unlikely that the preference structure is known apriori for human or other robotic opponents in a real-world setting.

The final challenge for autonomous racing algorithms is their real-time feasibility. Especially iterative trajectory optimiza-

tion methods, such as game-theoretic planners are limited by the amount of computation that can be done for real-time control. The computational constraints of autonomous racing are somewhat unique: The better autonomous racing algorithms become, the faster and more aggressively the vehicles will race over the track. This requires algorithms to become more competitive and less computationally intensive at the same time, two requirements that are somewhat in tension with each other.

II. Contributions and Organisation

These four challenges make the multi-agent autonomous racing problem highly complex and obviously intractable. Therefore we think it is unlikely that practical algorithms with provable safety and performance guarantees can be developed for the most general problem setting. Instead, we propose taking a data-driven approach by dividing the autonomous racing task, and hence our policies, into several different components. Application of the learning model predictive control (LMPC) algorithm [1] to autonomous racing has demonstrated great promise in learning locally optimal single-vehicle autonomous racing policies online in combination with online model learning strategies [2], and we hypothesize we can extend these successes to multi-agent settings by encoding effective long-term strategic decision making in the value-function approximation.

Although at a first glance it seems impossible for trajectory forecasting methods to allow for intelligent reasoning about the effect of the control policy's decisions on other agents, we argue that it is possible to accurately learn to predict the responses of opponents if we learn their responses in a competitive environment online, provided updates to the ego policy are small. This is because directly learning how the responses change online, in tandem with policy learning, mitigates the impacts of distributional shifts induced by updating the control policy. This allows us to avoid making assumptions on the preferences of the other agents and using costly game-theoretic iterative best-response algorithms.

This paper is structured as follows: We discuss related work in section §III and introduce our problem formulation in section §IV. We then introduce a trajectory forecasting method and provide a high-level, non-rigorous argument under what conditions we can rely on the learned forecasts in §V. Then, we present an simple adaptation of the LMPC algorithm [1] to a multi-agent setting in §VI. We outline our experiment in §VII and discuss the results in §VIII. We conclude and discuss future work in §IX.

III. Related Work

The learning model predictive control algorithm (LMPC) [1] is a predictive control algorithm that uses data from previous control iterations to learn optimal policies with safety guarantees in the form of persistent constraint satisfaction. The difference with regular MPC strategies is that in LMPC the cost-to-go, often referred to as the value function, is approximated based on data from previous trials. Application of LMPC in tandem with an online system-ID strategy based on local linear regression was shown to learn highly competitive policies for a single-agent racing task at the handling limits of the vehicle [2]. This was accomplished by updating the value function in a way that made the LMPC problem equivalent to a minimum-time control formulation [9]. The LMPC algorithm has been adapted to a multi-agent setting before [7], but the iterative response trajectory optimization scheme only allowed for a single response iteration to be completed in real-time, limiting its performance. This motivates our data-driven approach, avoiding the need for sequential algorithms.

In game theory, all the agents are assumed to act rationally towards some cost function. Given this knowledge, it becomes possible to anticipate how the other opponents will respond to decisions from any given agent. The agents are said to be at a Nash-equilibrium if it is in none of the agents' interest to change their behavior given that the others keep their policy the same. Since Nash-equilibria result in a stable interaction, they are desirable policies to search for in a competitive robotic setting. Although solving for Nash-equilibria is generally intractable, recent work on autonomous racing in drones [5] applied an iterative best-response algorithm and used sensitivity analysis to get real-time control performance. However, the preferences of the other agent were assumed as known. Other work in autonomous driving [8] used inverse reinforcement learning strategies [10] to learn the incentive structures of other road users and applied a simpler iterative response optimization scheme to approximate an equilibrium solution. However, inverse RL methods in this application [8] did not consider the confidence or uncertainty in the learned reward functions, thereby making it unclear if the computed equilibrium policies are also Nash-equilibria for the true multi-agent interaction. In addition, it is often unclear if humans act rationally with respect to a cost function in some hypothesis class, or if other agents have access to the same information as the ego agent to make their decission. Moreover, it is not obvious how one would propagate uncertainty or solve for robust nash-equilibria for all reward functions in some confidence set. Therefore, we simply consider the uncertainty on trajectory forecasts to reduce the likelihood of collisions.

Some approaches to the autonomous racing problem have focused more on robust control under model uncertainty, and emphasize learning optimal trajectories to a lesser extent. For example, Hewing et al. used Bayesian regression to learn an additive nonlinear component of the dynamics online for robust control, and used an offline trajectory optimizer to find a reference trajectory for the nominal model using a contouring cost [3]. Although this achieves good performance in a controlled experiment, the reference trajectory cannot be adapted online once the uncertainty in the dynamics is resolved. A similar strategy is employed by Liniger et al., who considered a multi agent racing problem [6]. The authors use a shortest paths algorithm online as a heuristic to decide how to overtake other agents, enforced by setting constraints on an MPC solver. Most related to our approach is recent work that takes a reinforcement learning approach [4]. The authors

synthesise a large set of decently performing policies that select waypoints through self-play in a simulated environment, after which the best policy is deployed. The waypoints are tracked using a sampling based trajectory optimizer, and the authors use an online learning algorithm to find a mixture of candidate policies to match the behavior of an opponent online. Although the authors do not investigate the ability of their online learning approach to generalize to agents not in the synthesized policy set and the experimental competitiveness of the method is low, this method suggests that learning to predict opponents' behavior through online trajectory forecasting is possible if the model is learned during competitive play.

IV. Problem Formulation

We consider the discrete time control of a system formed by the independent dynamics of two agents, the ego (agent 1) and the opponent (agent 2). For simplicity, we assume the dynamics of both agents occupy the same state space. Let $x_t^1, x_t^2 \in \mathbb{R}^n$ represent the states of the ego and the opponent at time t respectively, and let $u_t^1, u_t^2 \in \mathbb{R}^m$ be their respective inputs. We write the *known* joint dynamics with joint state $x_t = [x_t^{1\top}, x_t^{2\top}]^\top$, as

$$x_{t+1} = f(x_t, u_t^1, u_t^2) = [f_1(x_t^1, u_1^t)^\top, f_2(x_t^2, u_t^2)^\top]^\top, \quad (1)$$

where $f_i : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ represents the independent dynamics of the agents. In addition, both agents are subject to state and input constraints:

$$x_t^i \in \mathcal{X} \subseteq \mathbb{R}^n,\ u_t^i \in \mathcal{U} \subseteq \mathbb{R}^m\ \forall t \geq 0,\ i = 1, 2 \quad (2)$$

We assume access to a known *safety margin* $\mathcal{A} \subset \mathcal{X}$ that defines whether the two agents are in a collision or not. We say that the agents are in a collision state if and only if $x_t^1 \in x_t^2 \oplus \mathcal{A}$, where $\oplus$ is the Minkowski sum operator. This definition reflects the fact that the states of the system represent the physical positions of vehicles that have volume. The design goal of this work is to construct a policy $\pi_1 : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^m$ for the ego agent to safely achieve a competitive objective. We assume that both agents can perfectly observe the state of the system. In addition, we make some minimal assumptions on the behavior of the opponent. In particular, we assume that the opponent behaves according to a *consistent policy*:

$$u_t^2 = \pi_2(x_t^1, x_t^2) \quad \forall t \geq 0 \quad (3)$$

However, the opponent's policy is unknown, we only assume that it is fixed in time. The ultimate objective of the control task for agent 1 is to minimize the following infinite time optimal control problem

$$\begin{aligned} J_1^\star(x) = \min_{u_t^1} \sum_{t=0}^{\infty} & h(x_t, u_t^1) \quad (4) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t^1, \pi_2(x_t)) \quad \forall t \geq 0 \\ & x_t^1 \in \mathcal{X}, \quad u_t^1 \in \mathcal{U} \qquad \forall t \geq 0 \\ & x_t^1 \notin x_t^2 \oplus \mathcal{A} \qquad \forall t \geq 0 \\ & x_0 = x \end{aligned}$$

Unfortunately, the objective (4) is intractable for many reasons: The horizon is infinite, and since the dynamics are nonlinear and the collision avoidance constraints are not convex the problem (4) is non-convex. Moreover, since the policy of the opponent is unknown, we cannot hope to solve (4). Instead, we take an iterative learning approach and attempt the task episodically from the same initial condition and solve a finite horizon approximation of (4) using Learning MPC [1] online. By iteratively attempting the task, the controller should improve its performance over time. At the j'th iteration of the control task, we therefore assume access to trajectory data of the previous iterations $\tau_k = \{\{x_{k,0}, u_{k,0}^1, u_{k,0}^2\}, \ldots, \{x_{k,T_k}, u_{k,T_k}^1, u_{k,T_k}^2\}\}$ for $k = 0, \ldots, j-1$. Here, we use the double subscript $x_{k,t}$ to refer to the state recorded at time t of the k'th iteration of T_k timesteps.

V. Trajectory Forecasting

In practice, learning an opponents' policy is challenging, as perfect state and input information may not be shared between the agents. In addition, the dynamics of the opponent may not be identical to the dynamics of the ego vehicle. We therefore do not construct an estimate of the opponent's exact policy $\widehat{\pi}_2(x_t)$ in this work. Instead, we simply learn to predict the next N states of the opponent's dynamics:

$$\hat{X}_t^2 := [\hat{x}_{t+1|t}^2, \ldots, \hat{x}_{t+N|t}^2] = h_\theta(x_t) \quad (5)$$

Here, $\hat{x}_{t+i|t}^2$ refers to the prediction of the state of agent 2 at time $t+i$, made at time t using the learned function $h_\theta(\cdot)$ parametrized by θ. However, the joint state space is usually too high-dimensional to properly learn the opponents' policy from a handful of trajectories that only visit a small subset of the state space. For a safety critical control task, acting on poor predictions of an opponent's behavior can lead to dangerous situations and should be avoided.

This is particularly important because the opponent acts based on the joint state of the system, and therefore the ego agent's future decisions will affect the validity of the prediction (5). Via a heuristic argument, we consider this as a problem of learning under a distribution shift. Even though we do not explicitly consider noise in the problem formulation 1, it is in general the case that if we fix the policy of the ego vehicle $\pi_1(x_t)$ and run the control task to collect data, the control policy will induce a distribution over closed loop trajectories $\tau \sim P_{\pi_1}(\tau)$. Therefore, under a fixed ego policy, we should be able to efficiently learn the opponent's behavior over the induced trajectory distribution. This is particularly easy if the dynamics are deterministic, since in this case a fixed ego policy and a consistent opponent will result in the same system trajectory every time the control task is attempted. The difficulty arises when we update the ego policy between iteration j and $j+1$, since this results in a shift in the distribution over trajectories $P_{\pi_1^j}(\tau) \to P_{\pi_1^{j+1}}(\tau)$ that requires us to extrapolate the estimated behavior of the opponent to states outside of the domain observed previously. With some abuse of notation, if we assess the performance of the learned trajectory forecast $h_\theta(x_t)$ using a loss criterion

$\ell(h_\theta(x_t), \pi_2(\cdot))$ and define the expected loss induced by the trajectory distribution $P_{\pi_1^j}(\tau)$ as

$$L_{\pi_1^j}(\theta) = \mathbb{E}_{P_{\pi_1^j}}[\ell(h_\theta(\cdot), \pi_2(\cdot))], \tag{6}$$

then we can write the expected loss of the estimate on the trajectory distribution induced by the updated policy $\pi_1^{j+1}(\cdot)$ as

$$L_{\pi_1^{j+1}}(\theta) = L_{\pi_1^j}(\theta) + [L_{\pi_1^{j+1}}(\theta) - L_{\pi_1^j}(\theta)]. \tag{7}$$

Equation (7) makes it clear that if we ensure the distributional shift over trajectories between the ego policies at iteration j and $j+1$ is small, then we can expect the difference term in (7) to be small, therefore implying that our estimate of the opponent's behavior will be accurate for the updated policy as well. As a result, we propose a control algorithm for which the policy update after each iteration j is small as a proxy to ensure the trajectory distribution is small. In other words, as long as we make incremental updates to the ego policy $\pi_1^j(\cdot) \approx \pi_1^{j+1}(\cdot)$, then we can likely learn the opponent's behavior by play (or in competition) without creating dangerous situations.

VI. Learning Model Predictive Control

The Learning Model Predictive Control (LMPC) algorithm is precisely a policy iteration method that makes incremental updates between control iterations [1]. We start with the following observation: Since we assume the opponent acts consistently according to a fixed policy on the joint state, the system dynamics evolve only as a function of the initial condition and the decisions that the ego vehicle makes, since we can write $x_{t+1} = \hat{f}(x_t, u_t^1) = f(x_t, u_t^1, \pi_2(x_t))$. Therefore, this allows us to apply the existing Learning Model Predictive Control algorithm [1] to iteratively improve a finite horizon approximation of the ego vehicle's objective (4). We apply a formulation almost identical to [2]. The LMPC algorithm relies on value function approximation and a learned terminal constraint to guarantee recursive constraint satisfaction, asymptotic stability, and performance improvement. For nonlinear systems, retaining these guarantees requires mixed integer programming [1]. If we convexify the problem, these safety guarantees only hold for linear systems. However, the local convex approximations that we make have shown good performance in practice [2].

At iteration j, we define the cost-to-go samples for all previously recorded states for iterations $k = 0, \dots, j-1$ and timesteps $t = 0, \dots, T_k$ as

$$q_{k,t} = \sum_{i=t}^{T_k} h(x_{k,i}, u_{k,i}^1) \tag{8}$$

We then define the hyperparameter n_{ss} as a positive integer that determines how many sampled states are used to construct the value function approximation. At a query state x, we define the timestep for which the sampled trajectory k is nearest to the query state as

$$t_k(x) = \arg\min_{t \in [0, T_k]} \|x - x_{k,t}\|_2^2 \tag{9}$$

then, we define the *local safe-set* around a query state x for the sampled trajectory k as

$$\mathcal{SS}_k(x) = \bigcup_{i=-n_{ss}}^{n_{ss}} x_{k, i+t_k(x)} \tag{10}$$

The local safe-set for trajectory k essentially collects $2n_{ss}$ contiguous states centered around the sampled state nearest to the query point. For presentational clarity, we ignore edge cases around the start and end state of the trajectory. We then take the local safe-set around a query point x at control iteration j as

$$\mathcal{SS}^j(x) = \bigcup_{k=0}^{j-1} \mathcal{SS}_k(x), \tag{11}$$

and the *convex local safe-set* at iteration j around query point x as

$$\mathcal{CS}^j(x) = \text{conv}(\mathcal{SS}^j(x)) \tag{12}$$

We will use (12) as a data driven terminal constraint. In addition, we define the value-function approximation $Q^j : \mathcal{CS}^j \mapsto \mathbb{R}_+$ at iteration j associated with a local convex safe-set as the Barycentric interpolation of the cost-to-go samples (8) associated with the trajectory samples included in the convex safe-set (12). To do this, we define the set

$$\mathcal{QS}^j(x) = \{q_{k,t} : q_{k,t} \in \mathcal{SS}^j(x)\} \tag{13}$$

with cardinality $|\mathcal{QS}^j(x)| = |\mathcal{SS}^j(x)| = 2jn_{ss}$. With a slight abuse of notation, we use the shorthand x_i, q_i to denote the entries of $\mathcal{SS}^j$, $\mathcal{QS}^j$ respectively. The value function approximation at a query point x is then defined as the Barycentric interpolation using the following LP:

$$\begin{aligned} Q^j(x) = \min_{\lambda_i \geq 0} & \sum_{i=1}^{2jn_{ss}} \lambda_i q_i \\ \text{s.t.} \quad & \sum_{i=1}^{2jn_{ss}} \lambda_i = 1, \ \sum_{i=1}^{2jn_{ss}} \lambda_i x_i = x \end{aligned} \tag{14}$$

We then write the Learning MPC problem we attempt to solve at time t of iteration j using the trajectory forecast $h_\theta(x_t) = [\hat{x}_{t+1|t}^2, \dots \hat{x}_{t+N|t}^2]$ as the optimization

$$\begin{aligned} J^j(x_{j,t}) = \min_{u_{t+i|t}^1} & \sum_{t=0}^{N-1} h(x_{t+i|t}, u_{t+i|t}^1) + Q^j(x_{t+N|t}) \\ \text{s.t.} \quad & x_{t+i+1|t}^1 = f_1(x_{t+i|t}^1, u_{t+i|t}^1) \\ & x_{t+i|t} = [x_{t+i|t}^{1\top}, \hat{x}_{t+i|t}^{2\top}]^\top \\ & x_{t+i|t}^1 \in \mathcal{X}, \quad u_{t+i|t}^1 \in \mathcal{U} \\ & x_{t+N|t} \in \mathcal{CS}^j(x_{t+n|t}) \\ & x_{t+i|t}^1 \notin \hat{x}_{t+i|t}^2 \oplus \mathcal{A} \\ & x_{t|t} = x_{j,t} \end{aligned} \tag{15}$$

The LMPC problem (15) depends only on the current state and uses the convex local safe-set and value function approximations to improve performance using recorded data. In addition,

problem (15) uses the independence of the dynamics (1) and the trajectory forecast (5) to immediately optimize the ego vehicle's trajectory instead of reasoning about the response of the opponent to the actions of the ego. We emphasize that (15) is still non-convex, therefore we discuss some implementation details in §VII. Let the solution of (15) be $[u_{t|t}^{1,\star}, \dots, u_{t+N|t}^{1,\star}]$. We then apply the first input of the LMPC trajectory to the system at time t to close the loop:

$$u_t^1 = u_{t|t}^{1,\star} \tag{16}$$

In practice, under the right selection of hyperparameters, the closed loop system formed by (1), (15), (16) typically converges to a locally optimal trajectory for the objective (4) [2]. This is based on the assumptions that the LMPC problem (15) ensures collision avoidance, and ultimately cannot improve its trajectory anymore. Practically, this means that the learned trajectory forecaster (5) and the LMPC policy (16) converge to an equilibrium trajectory.

VII. Experiment Details and Heuristics

To illustrate our approach, we simulate a *single-lap* race between two autonomous vehicles. We simulate the vehicle dynamics (1) as two identical dynamic bicycle models with global state $x_t^i = [x, y, \psi, v_x, v_y, \dot{\psi}]$, where x, y indicate the position of the vehicle, ψ its heading, and v_x, v_y its longitudinal and lateral velocities. The bicycle model has two inputs $u_t^i = [a, \delta]$, the longitudinal acceleration and steering angle. The continuous-time dynamics are given as:

$$\dot{x} = \cos(\psi)v_x - \sin(\psi)v_y \tag{17}$$

$$\dot{y} = \sin(\psi)v_x - \cos(\psi)v_y \tag{18}$$

$$\dot{v}_x = \dot{\psi}v_y + a - \frac{1}{m}F_{cf}\sin(\delta) \tag{19}$$

$$\dot{v}_y = -\dot{\psi}v_x + \frac{1}{m}(F_{cf}\cos(\delta) + F_{cr}) \tag{20}$$

$$\ddot{\psi} = \frac{l}{I}(F_{cf}\cos(\delta) - F_{cr}) \tag{21}$$

With mass $m = 2kg$, inertia $I = .03kg\ m^2$ and length $l = .125m$. The front and rear cornering forces are functions of the tyre slip angles and stiffness constant $c = 46N/rad$, $F_{cf} = -c\alpha_f$, $F_{cr} = -c\alpha_r$, with $\alpha_f = \arctan((v_y + l\dot{\psi})/|v_x|) - \delta$ and $\alpha_r = \arctan((v_y + l\dot{\psi})/|v_x|)$. As in [2], we transform the global vehicle state position and orientation into a curvilinear reference frame along the centerline of the track. As illustrated in Fig. 1, this allows us to rewrite the state as $x_t^i = [s, e_y, e_\psi, v_x, v_y, \dot{\psi}]$ with s as the distance along the centerline of the track and e_y, e_ψ as the tracking and orientation error. We simulate the system at a timestep of $.1s$ using a 4th order Runge Kutta solver.

For the opponent's policy, we take a simple MPC that follows the centerline of the racetrack at a fixed reference speed using a linearized dynamics model along the centerline. The initial condition of the race is identical at each iteration: The ego vehicle starts at the center of the starting line, and the opponent is given a fixed head start. The ego vehicle updates its performance using the LMPC policy formed by (15), (16).

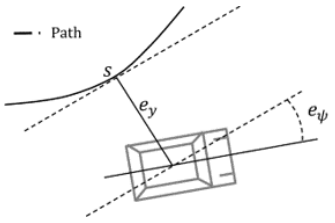

Fig. 1. Curvilinear reference frame along track centerline.

As a heuristic for a "win the race" objective, we take the objective of the LMPC to minimize the time to complete a lap using the the objective $h(x_t, u_t^1) = \mathbb{1}\{s > \text{tracklength}\}$ where $\mathcal{X}_f$ is the set of all states across the finish line. We also added a penalty on the rate of change of the inputs. We set a speed limit of 2 m/s, restricted steering between $\pm\pi/6$, and set the acceleration limit to $.5m/s^2$. We also added constraints associated with the track boundaries. We set the prediction horizon to $N = 10$. To initialize the simulation, we record 3 trajectories in which both agents use the simple lane-keeping MPC. This allows us to initialize the safe-set (12) and value function approximation (14) with some initial data. We set $n_{ss} = 20$ and only use the fastest 3 trajectories from previous iterations to construct the safe-sets and value-function approximations (12), (14). We learn the trajectory forecasting function $h_\theta(\cdot)$ using Gaussian Process regression [11].

To solve the LMPC problem (15), we use a heuristic method to compute the constraint tightening associated with the collision avoidance constraint $x_{t+i|t}^1 \notin \hat{x}_{t+i|t}^2 \oplus \mathcal{A}$ in (15). We make the constraint tightening active if the vehicle positions $|s^1 - s^2| \leq 2m$ and use the curvature of the road to decide if an overtake should be made to the left or right. If the curvature $\kappa(s) \geq 0$, this indicates a left-turn, and we tighten the constraints to force a left overtake. If the curvature is negative, this indicates a right turn, and we tighten the constraints to force the ego vehicle to keep a safe distance to the right of the opponent. We then solve the tightened LMPC problem (15) using a single iteration of Sequential Convex Programming (SCP). We initialize the SCP iteration using the previous timestep's solution and a future state predicted using the convex safe set and value function approximation as in [2]. We found that using this initialization, solving multiple SCP iterations did not lead to significant performance increases. We implemented our algorithm in Python using CVXPY and the OSQP solver [12]. To increase the performance, we build the solvers once using parameter objects and update the parameter values online without rebuilding the solvers. This allowed the simulation to run at about 3x real-time speed on a standard laptop.

VIII. Results

We ran the experiment for a total of 10 laps after collecting the 3 initialization laps. No collisions occurred during any of the laps. The opponent starts with a significant advantage over the LMPC policy, but after 10 laps the ego agent has efficiently learned to overtake the opponent and win the race. The

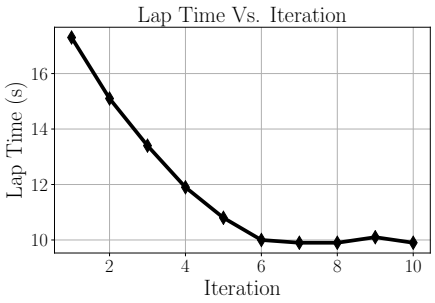

Fig. 2. Lap time per iteration for the LMPC policy.

trajectory forecasts are accurate throughout the experement, although our task is obviously somewhat simplified since we learn to forecast the trajectory of a fixed MPC policy. In addition, it seems like the heuristic overtaking method effectively avoids collisions. As shown in Fig. 2, the LMPC policy rapidly improves, reducing the lap time by about 7 seconds from the initial trajectory. Fig. 3 shows how the ego vehicle's velocity profile increases as more iterations are completed. Fig. 4 shows that the ego vehicle learns a locally optimal racing line around the L-shaped racetrack. The ego starts near the bottom right of the track, around the $x = 2$ mark. The opponent starts on the other side of the semicircular curve at about $x = 2,\ y = 2$. The agents complete the labs counterclockwise.

Finally, Fig. 5 shows how the ego vehicle learns to overtake the opponent on the second lap. During the previous iteration the LMPC's collision avoidance constraints were active at the corner, forcing the LMPC to take an inside line. On the second lap, the LMPC knows to plan more efficiently into the safe set associated with the inside line overtake. Still, the collision avoidance constraints from the trajectory forecast are active, resulting in a small kink in the predicted trajectory to guarantee the ego doesn't collide with the opponent. As the LMPC learns to drive faster around the track, the point at which the ego overtakes the opponent becomes closer to the starting line. On the final trajectory, the ego overtakes the opponent around the apex of the turn at the top of the L-shaped track (about

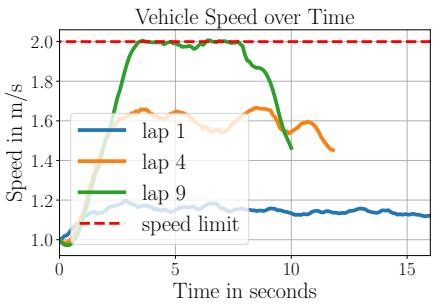

Fig. 3. Ego vehicle speed over time for several iterations.

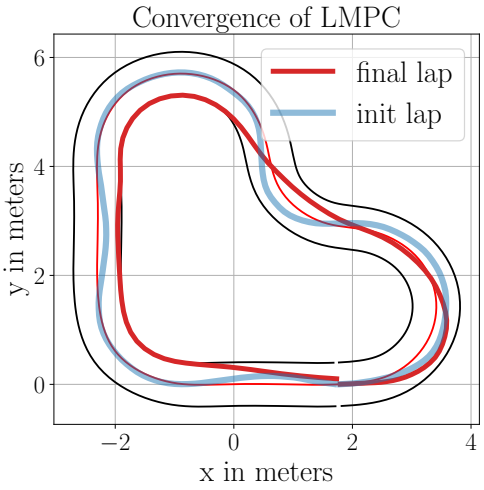

Fig. 4. Initial and learned trajectories of the LMPC. The track limits and center-line are displayed in black and red.

$x = -1,\ y = 5$ in Fig. 4).

IX. Conclusion and Future Work

In this work, we presented a data-driven method for multi-vehicle autonomous racing. We propose to learn a competitive policy by interacting with an opponent during competition and learning its behavior. We construct a predictive control algorithm that we observed to be safe during the learning process. Under an assumption that an opponent acts consistently with respect to the joint system state, we used learning methods to forecast the evolution of the opponent's state online. We applied a Learning MPC to iteratively update a data-driven value function approximation and terminal constraint to learn

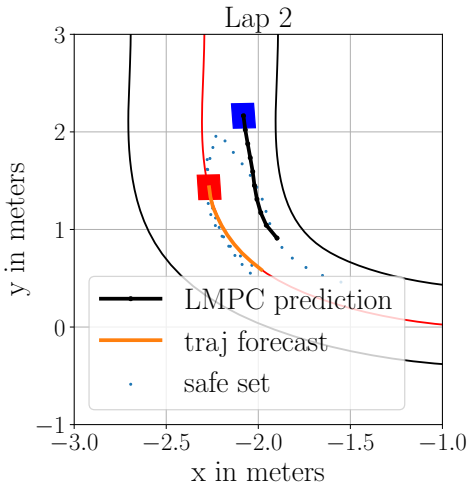

Fig. 5. The LMPC going for an overtake on lap 2. The ego vehicle is shown in blue, the opponent in red. The track limits and center-line are displayed in black and red.

a competitive policy, and edited the constraints of the LMPC to avoid collisions along the prediction horizon online using trajectory forecasts. We argued that the incremental policy updates of the LMPC reduce the distribution shifts that can impact the accuracy of the trajectory forecasts, thereby increasing the reliability of the predictions. Simulated experiments against a simple MPC opponent showcased how this approach was rapidly able to learn to win the race.

Although our results are encouraging, we were unable to simulate our algorithm against agents that exhibit more competitive blocking and overtaking behavior. Therefore, future work should further test and benchmark our approach to accurately quantify the performance of our methods against more general competitors. Furthermore, we use a heuristic method to decide between left and right overtakes. In future work, it would be interesting to examine whether this strategic decision making can be learned as well. Moreover, planning around (Bayesian) confidence intervals on the trajectory forecasts could further improve the safety of our approach. Finally, our method can currently only learn the behavior of a single opponent by competitively interacting with it. Future work should investigate learning methods that can be used to generalize behaviors of other agents and quickly adapt to new opponents, for example by using meta-learning [13].

REFERENCES

[1] Ugo Rosolia and Francesco Borrelli. Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, July 2018. Conference Name: IEEE Transactions on Automatic Control.

[2] U. Rosolia and F. Borrelli. Learning How to Autonomously Race a Car: A Predictive Control Approach. *IEEE Transactions on Control Systems Technology*, 28(6):2713–2719, November 2020. Conference Name: IEEE Transactions on Control Systems Technology.

[3] Lukas Hewing, Juraj Kabzan, and Melanie N. Zeilinger. Cautious Model Predictive Control using Gaussian Process Regression. May 2017.

[4] Aman Sinha, Matthew O'Kelly, Hongrui Zheng, Rahul Mangharam, John Duchi, and Russ Tedrake. FormulaZero: Distributionally Robust Online Adaptation via Offline Population Synthesis. In *International Conference on Machine Learning*, pages 8992–9004. PMLR, November 2020. ISSN: 2640-3498.

[5] Riccardo Spica, Davide Falanga, Eric Cristofalo, Eduardo Montijano, Davide Scaramuzza, and Mac Schwager. A Real-Time Game Theoretic Planner for Autonomous Two-Player Drone Racing. *IEEE Transactions on Robotics*, 36(5), 2020.

[6] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–647, 2015. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/oca.2123.

[7] Lukas Brunke. Learning Model Predictive Control for Competitive Autonomous Racing. *arXiv:2005.00826 [cs, math, stat]*, May 2020. arXiv: 2005.00826.

[8] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for Autonomous Cars that Leverage Effects on Human Actions. page 9.

[9] Ugo Rosolia and Francesco Borrelli. Minimum time learning model predictive control. *International Journal of Robust and Nonlinear Control*, n/a(n/a).

[10] Andrew Y. Ng and Stuart Russell. Algorithms for Inverse Reinforcement Learning. In *in Proc. 17th International Conf. on Machine Learning*, pages 663–670. Morgan Kaufmann, 2000.

[11] Carl Edward Rasmussen, Christopher K. I. Williams, and Francis Bach. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[12] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[13] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning Priors for Efficient Online Bayesian Regression. In Marco Morales, Lydia Tapia, Gildardo Sánchez-Ante, and Seth Hutchinson, editors, *Algorithmic Foundations of Robotics XIII*, Springer Proceedings in Advanced Robotics, pages 318–337, Cham, 2020. Springer International Publishing.