2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 1-5, 2023. Detroit, USA

Kinematics-Only Differential Flatness based Trajectory Tracking for Autonomous Racing

Yashom Dighe, Youngjin Kim, Smit Rajguru, Yash Turkar,
Tarunraj Singh, Karthik Dantu

2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-9190-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/IROS55552.2023.10341603

***Abstract*— In autonomous racing, accurately tracking the race line at the limits of handling is essential to guarantee competitiveness. In this study, we show the effectiveness of Differential Flatness based control for high-speed trajectory tracking for car-like robots. We compare the tracking performance of our controller against Nonlinear Model Predictive Control and resource use while running on embedded hardware and show that on average KFC reduces the computation resource usage by 50 % while performing on par with NMPC. Our implementation of the proposed controller, the simulation environment and detailed results is open-sourced on https://github.com/droneslab/.**

I. INTRODUCTION

Recently, there has been a lot of enthusiasm in the research [1] and hobby [2] community in autonomous racing. This is particularly attractive given these platforms are affordable (a few hundred to a few thousand US dollars) and easy to build. This has also led to the rise of several autonomous racing competitions including the F1tenth [1] racing at academic and hobby venues. This application provides an opportunity to revisit previously studied research ideas in other contexts for autonomous racing where time optimality requires aggressive maneuvers. Achieving the fastest lap times requires a race line (time optimal trajectory across the track) and a tracker to follow it perfectly. A perfect race line is useless without a tracker that can track it well. In this work, we employ the idea of differential flatness [3] and its use in tracking a pre-planned trajectory.

We start with a bicycle model, which has been previously studied to replicate the dynamics of the vehicle or car-like robot [4],[5],[6]. Additionally, it is proven to be differentially flat [7] and is known to be feedback linearizable i.e. the error dynamics are linear. This lets us apply simple linear feedback control and guarantee asymptotic stability. Leveraging this fact, we propose a simple PD controller based on this linear feedback and compare its performance against Nonlinear Model Predictive Control (or Non-linear MPC) which is the state of the art for racing [8], [9], [10]. As effective as MPC is, it is also quite computationally intensive because it solves an optimization problem at every time-step. While modern micro-controllers and single-board-computers (SBCs) have come quite far, running complete autonomy pipelines on them still stretches their computational budget. This becomes more evident when it comes to racing because all aspects of the pipeline - perception, state estimation, planning and control have to be executed at extremely high rates (tens of Hz) making a lightweight controller desirable. To that end, we make the following contributions in this work:

Fig. 1: Sample Track for F1tenth Autonomous Racing experiments

- Using the bicycle model kinematic approximation of a car, we develop a trajectory tracker that leverages the flatness property of the said model - Kinematic Flat Controller (KFC)
- Using four tracks in simulation as well as three in the real world, we compare the tracking performance of the proposed controller against a Nonlinear Model Predictive Controller (NMPC) for high speed trajectories
- In the process, we made adaptations to the F1tenth hardware platform as well as the simulator [11] to fit our model. All of this work is open-sourced on our group GitHub page[1]

II. RELATED WORK

The idea of differential flatness introduced by [12] is well recognized for global trajectory/motion planning problems. Various robotic systems such as helicopters [13], fixed-wing air-crafts [14], quad-rotors [15], and unicycle type robots [16] are considered to be differentially flat. Considerable work has been done when it comes to the control of car-like robots in flat space - [17] have used the concept of differential flatness to apply trajectories computed for the unicycle model to the bicycle model, [18] use flatness properties to drive on structured motorways, [19] impose constraints on the flat space to guarantee safety in the state space, [20] map tire friction forces to constraints in the

[1]All authors are with University at Buffalo, NY, 14260 USA {yashomna,ykim35,smitmaka,yashturk,tsingh,kdantu}@buffalo.edu
Contact author is Yashom Dighe (yashomna@buffalo.edu)

[1]https://github.com/droneslab

flat space to solve the local steering problem, [21] leverage the differential flatness property to simplify the trajectory planning problem. However, the idea has not been evaluated in the context of tracking aggressive trajectories needed for racing (we refer the reader to the work of [22] for a survey of autonomous racing). [23] and [24] show the effectiveness of differential flatness in tracking aggressive quad-rotor trajectories and [25] do an extensive study of NMPC vs Differential Flatness based trajectory tracking for quad-rotors. But, key differences between the quad-rotor and bicycle models such as having only 3 DOFs compared to 6 and the presence of non-holonomic constraints motivate us to re-investigate for car-like (ackermann steered) robots.

III. IMPLEMENTATION

This section describes generation of the reference trajectory, and the designs of Kinematic Flat Controller (KFC) and the Non-linear Model Predictive Controller (NMPC).

A. Bicycle Model Kinematics

The kinematic model of a simplified car is derived under the assumption that the vehicle is rear-wheel driven, front wheel steer and wheels have no slip.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} V\cos(\theta) \\ V\sin(\theta) \\ \frac{V}{L}\tan(\delta) \end{bmatrix} \tag{1}$$

where (x, y) represent the position of the center of the rear axle, θ is the orientation of the vehicle, and the length of the wheelbase is L. The input to the system is the forward linear velocity V, steering angle δ.

B. Trajectory Generation

A trajectory is a time parameterized path along with a velocity profile. We use cubic splines to parameterize a given set of waypoints with a fixed length time vector. The derivative of this spline gives the velocity profile. To fit the spline we use scipy [26]

Through simulation and experimentation, we classify generated trajectories into two classes — *feasible* and *infeasible*. Feasible trajectories are ones that obey the limits of the system dynamics and represent a practical racing scenario. Infeasible trajectories are characterized by velocity profiles that are not permissible under system dynamics even though they are kinematically possible. These include scenarios such as trying to turn with speeds that are not permissible under the centripetal forces experienced by the car. These trajectories represent a real world worst case scenario for the controllers. They let us test the controllers' robustness to model mismatch, external disturbances such as tire slip and actuator saturation. Further, racing requires operating at the limits of handling where even the smallest model mismatch or disturbance can render a feasible trajectory infeasible. In the following subsections we describe the method used to generate trajectories of each type.

1) Feasible Trajectories: Dynamically feasible trajectories require slowing down on the turns. To achieve this, we compute a time vector that scales the time between waypoints based on a pre-calculated mapping between instantaneous curvature (κ) at that waypoint (calculated using the *first curvature* [27], [28]) and the max permissible speed at that curvature. To calculate this mapping, we command the car with various velocities at maximum steering angle and measure the curvature of the traversed arc. The data points gathered from these experiments draw a hyperbolic arc in the V vs δ plane. Estimating the parameters of this hyperbola gives a function that maps the curvature κ to the maximum permissible speed $u_{max,\kappa}$.
For computing the time vector, we formulate a minimization problem constrained by a given max speed V_{max} and max acceleration a_{max} to generate a nearly time-optimal feasible trajectory for a given set of waypoints on the track

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{N} T_i \\ \text{subject to } & T_i = \sum_{r=0}^{i} \Delta t_r \\ & u_{i+1} = u_i + a_i \Delta t_i \\ & u_{min} \leq u_i \leq argmin(u_{max}, u_{max,\kappa}) \\ & |a_i| \leq a_{max} \\ & \Delta t_i = \frac{D_i}{u_i} \end{aligned}$$

where, u is the velocity, a is the acceleration, u_{max} is the maximum velocity, $u_{max,\kappa}$ is the maximum permitted velocity for curvature κ, u_{min} is the minimum velocity needed to overcome the inertia, D_i is the distance between the i^{th} and the $i^{th}+1$ waypoint and Δt_i is the time taken to traverse from the ith waypoint to the $i^{th}+1$ waypoint.

2) Infeasible Trajectories: The racetracks are a combination of straights that allow for extremely high speeds and turns which require the car to slow down to a near crawl in order to be able to negotiate these turns. To generate infeasible trajectories a linearly spaced time vector is fit to the waypoints of the tracks (using cubic splines as mentioned before) i.e., the reference trajectory consists of a uniform, nearly constant velocity profile that disregards the shape of the track. By changing the total amount of time taken to complete a lap, we can choose the average speed over the length of the track. This is a simple approach but as mentioned earlier, lets us evaluate the controllers in a worst case scenario. The cars are expected to overshoot on the turns but catch up to the reference on the straights.

C. Kinematic Flat Controller

A system is considered as differentially flat if there exists a set of outputs (flat outputs), equal to the number of inputs, such that all the states and controls in state space can be represented in terms of the selected flat outputs and their derivatives without integration. The bicycle model is differentially flat with positions as the flat outputs. Let f_1 and

f_2 be the flat outputs, which map to the position coordinates x and y as

$$\begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

Then, the one-to-one mapping between the state space model and flat space is given by

$$\begin{bmatrix} \dot{f}_1 \\ \dot{f}_2 \end{bmatrix} = \begin{bmatrix} V\cos(\theta) \\ V\sin(\theta) \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} \ddot{f}_1 \\ \ddot{f}_2 \end{bmatrix} = \begin{bmatrix} \dot{V}\cos(\theta) - V\dot{\theta}\sin(\theta) \\ \dot{V}\sin(\theta) + V\dot{\theta}\cos(\theta) \end{bmatrix} \tag{4}$$

Since $\theta = \arctan(\frac{\dot{y}}{\dot{x}})$,

$$\theta = \arctan\left(\frac{\dot{f}_2}{\dot{f}_1}\right) \tag{5}$$

Similarly, the forward linear velocity (V) and steering command (δ) can be rewritten as

$$V = \sqrt{\dot{x}^2 + \dot{y}^2} = \sqrt{\dot{f}_1{}^2 + \dot{f}_2{}^2} \tag{6}$$

$$\delta = \arctan\left(\frac{L\dot{\theta}}{V}\right) = \arctan\left(L\frac{\dot{f}_1\ddot{f}_2 - \dot{f}_2\ddot{f}_1}{(\dot{f}_1{}^2 + \dot{f}_2{}^2)^{\frac{3}{2}}}\right) \tag{7}$$

Let U_1 and U_2 be the virtual controls in flat space:

$$\ddot{f}_1 = U_1 \tag{8}$$

$$\ddot{f}_2 = U_2 \tag{9}$$

we define the linear feedback law in a Proportional-Derivative (PD) manner with a feed-forward term as

$$U_1 = \ddot{f}_1{}^d + K_1\dot{e}_{f_1} + K_2 e_{f_1} \tag{10}$$

$$U_2 = \ddot{f}_2{}^d + K_3\dot{e}_{f_2} + K_4 e_{f_2} \tag{11}$$

where superscript $(.)^d$ is reference trajectories, $e_{(.)} = f^d_{(.)} - f_{(.)}$ is error feedback and $K_1 - K_4$ are the control gains.

Rearranging the eqn. (10) and (11) yields

$$\ddot{e}_{f_1} + K_1\dot{e}_{f_1} + K_2 e_{f_1} = 0 \tag{12}$$

$$\ddot{e}_{f_2} + K_3\dot{e}_{f_2} + K_4 e_{f_2} = 0 \tag{13}$$

It can be seen that the error feedback dynamics are linear, and as long as control gains are positive the feedback errors asymptotically go to zero. Since the proposed feedback controls are designed in a flat space, they need to be mapped back to the coordinate space for real-time implementation. As derived earlier, the mapping from $[U_1, U_2]$ to flat space is given by

$$\begin{aligned} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} &= \begin{bmatrix} \dot{V}\cos(\theta) - V\dot{\theta}\sin(\theta) \\ \dot{V}\sin(\theta) + V\dot{\theta}\cos(\theta) \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \cos(\theta) & -V\sin(\theta) \\ \sin(\theta) & V\cos(\theta) \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \dot{V} \\ \dot{\theta} \end{bmatrix} \end{aligned} \tag{14}$$

Inverse of the matrix **A** leads to :

$$\dot{V} = U_1\cos(\theta) + U_2\sin(\theta) \tag{15}$$

$$\dot{\theta} = \frac{U_2\cos(\theta) - U_1\sin(\theta)}{V} \tag{16}$$

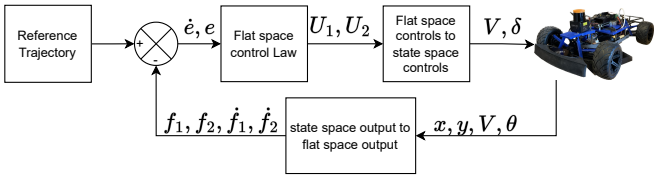


Fig. 2: KFC Feedback Control Block Diagram

Since the input to the system is V and δ

$$V = \int \dot{V}\, dt \tag{17}$$

$$\delta = \arctan\left(\frac{L}{V^2}(U_2\cos(\theta) - U_1\sin(\theta))\right) \tag{18}$$

Note that when V goes to zero, the matrix **A** becomes a singular matrix making δ to be undefined. To avoid this issue, we used a method from [29]. We define a threshold velocity V_T

$$V = \begin{cases} V_T & \text{when } |V| \leq V_T \\ V & \text{when } |V| > V_T \end{cases} \tag{19}$$

where V_T is a small positive value. One of the potential shortcomings of using this idea is that the car loses its ability to move backward. However, in the context of racing, backward maneuvers are not required. The proposed control architecture is summarized in Fig. 2.

D. Nonlinear Model Predictive Controller (NMPC)

To generate the control commands, NMPC solves a finite time optimal control problem (OCP) in a receding horizon fashion. For a given reference trajectory, the cost function is calculated using the error between the predicted states and the reference points in the time horizon. States and inputs are discretized into N equal intervals of the time horizon $T \in [t, t+h]$ with $dt = h/N$ where h denotes the horizon length. Our implementation of NMPC is based on [30]. Using the bicycle model kinematics to predict the states, we formulate a nonlinear optimization problem that is solved at every time step as follows

$$\begin{aligned} \text{minimize } & \sum_{j=0}^{N-1} u_j^T R u_j + \sum_{i=1}^{N} (X_{ref,i} - X_i)^T Q (X_{ref,i} - X_i) \\ \text{subject to } & X_{i+1} = f(X_i, u_i) \\ & |u_{j+1} - u_j| \leq [\Delta a_{max}, \Delta\delta_{max}]^T \\ & X_0 = X_{init} \\ & u \in [u_{min}, u_{max}]^T \end{aligned}$$

Where,
$R = diag(R_{acc}, R_{steer})$, is the penalty on the controller effort. $Q = diag(Q_x, Q_y, Q_\theta, Q_v)$, is the weight matrix and Q_x, Q_y, Q_θ, Q_v are the weights associated with the error for each state. $X_i = [x_i, y_i, \theta_i, v_i]$, (x, y) is position, θ is yaw, v is forward velocity. $u = [a, \delta]$, a is acceleration, δ is the steering angle (v is calculated from a using integration), $\Delta a_{max}, \Delta\delta_{max}$ are used to limit jerk and erratic steering,

$$f(x) = \begin{bmatrix} x_{i+1} \\ y_{i+1} \\ \theta_{i+1} \\ v_{i+1} \end{bmatrix} = \begin{bmatrix} x_i + v_i \cos(\theta_i) dt \\ y_i + v_i \sin(\theta_i) dt \\ \theta_i + \frac{v_i \tan(\delta_i) dt}{L} \\ v_i + a_i dt \end{bmatrix}$$

as per the bicycle model kinematics.

Solving this optimization problem yields $[a_i, \delta_i]$. We use the method described in eqn. (17) to obtain a $[v_i, \delta_i]$ which are the control inputs to the car.

IV. EVALUATION

This study aims to evaluate the competitiveness of differential flatness for autonomous car racing by answering the following questions

1) How does the position tracking performance of KFC compare to NMPC?
2) What is the computation effort needed for each controller?

Through extensive simulation as well as real-world experimentation across multiple trials, we demonstrate that the kinematics-based flatness controller is a superior approach and competitive compared to Non-linear MPC.

A. Experimental Setup

Simulation: The simulation setup uses Gazebo11, ROS Noetic, and builds upon the work of [11] to meet our requirements. Major changes include but are not limited to, remodeling the vehicle from all-wheel drive to rear-wheel drive as per bicycle model kinematics, modeling the dynamics of the simulated vehicle to match those of the actual car, increasing the speed limit and matching the steering limits to that of the real car. The simulation parameters (friction, actuation limits, maximum acceleration) were tuned to match its behavior to the real car. This was done by commanding the real and simulated car with an input and matching the resulting output through trial and error. Experiments were conducted on $1/10^{th}$ scale F1 tracks provided by F1tenth GitHub [31]. The tracks chosen were - IMS (292 m), Monza (446 m), Silverstone (457 m) and Oschersleben (260 m). As with the real-world experiments, multiple runs were conducted at each max speed per track

Experiment: The testing platform is built using a modified version of the hobby RC car chassis - Traxxas Slash 4x4. Modifications include converting it from all-wheel-drive to rear-wheel-drive, eliminating the suspension and lowering the center of gravity. Experiments were conducted on 3 tracks at different max speeds - a loop (20 m), a figure 8 pattern (30 m) and a U-shaped loop (30 m). Multiple runs were conducted at each max speed per track. All experiments were conducted in an environment with a VICON motion capture system for ground truth.

B. Metrics to Evaluate Tracking Performance

The tracking performance of each method is evaluated based on 3 metrics -

- Root Mean Square (RMS) of the position tracking error ($rmse_p$) calculated as the perpendicular distance between the nearest point of path to be tracked and the executed path. This metric is chosen because deviating from the optimal path can result in an increase in lap time.
- RMS of the trajectory tracking error ($rmse_t$) reveals how well the position is tracked in time i.e. where the car is supposed to be at that time instant vs where it actually is. This is crucial to racing because it lets us quantify the time sacrificed to achieve the position tracking accuracy.
- Maximum deviation from the centerline (ϵ): Because the controller is evaluated in the context of racing, we use this metric to measure if the car overshoots the track boundary. For infeasible trajectories, it also lets us record the worst overshoots on extremely tight turns.

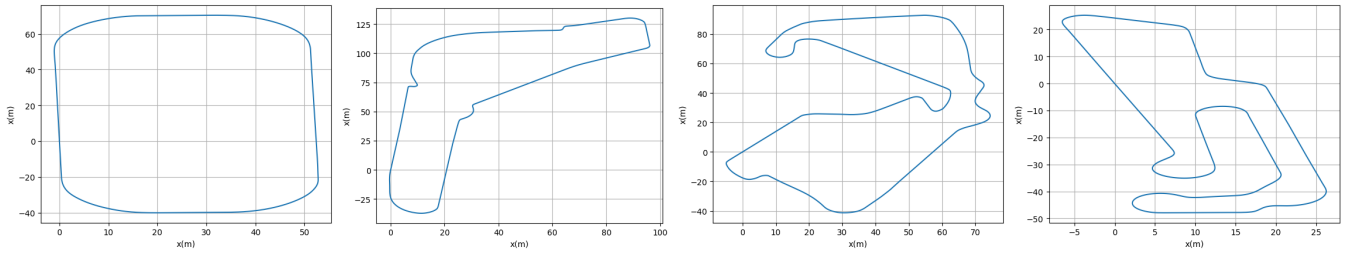
C. Measuring Resource Utilization

The controllers were implemented in Python using ROS Noetic on the Nvidia Jetson Xavier NX running jetpack 5.1. Across all the tests (simulation and real world) usage stats were logged using the process IDs of the controller process. The controllers were implemented as ROS nodes that were run on the Jetson and run at 100Hz. When running simulations, the rosmaster and the simulator were run on a different computer and the controller nodes connected to them over the network. For both the controllers all the cubic spline interpolation was done using SciPy [26] For NMPC, CasADi [32] was used to solve the nonlinear optimization problem.

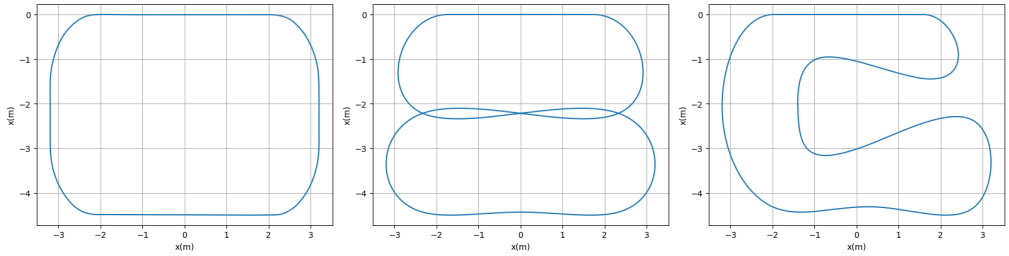
D. Simulation results

Given that the limit of the car is 15 m/s in the simulation, the maximum speed tested for is 12 m/s to let the controllers have some margin to operate within. Feasible trajectories for $V_{max} \in \{8, 10, 12\} m/s$ were tested across all tracks. These max speeds were chosen because this is where the controllers start failing for infeasible trajectories. For speeds lower than $8m/s$ feasible trajectories performed better than their infeasible counterparts and the performance of both controllers was similar. These results have not been tabulated to save space but will be made available on our GitHub. Table I presents the results for individual tracks for speeds $> 8m/s$. From the data, it can be seen that there is no clear winner and both methods perform similarly for all speeds when it comes to $rmse_t$. NMPC shows marginally better results for $rmse_p$ and ϵ. We note that despite these trajectories being feasible, both controllers struggle at high speeds as can been seen from large values for max. deviation especially at higher speeds on track 3d.

For infeasible trajectories, Table II presents the results for each individual track. On tracks 3c and 3d neither controller converges back on to the track in reasonable time once there is an overshoot for speeds higher than 7 m/s and no data is recorded. Conversely, we noticed that for track. 3a and track. 3b the results for 6 and $7m/s$ were nearly identical to the



(a) IMS (b) Monza (c) Silverstone (d) Oschersleben



(e) Loop (f) Figure 8 (g) U-Loop

Fig. 3: Center-lines of the race tracks tested on. Top row has tracks tested in simulation and bottom row has real world tracks

TABLE I: Simulation results for feasible trajectories

Track	Avg. Speed	Method											
		KFC						NMPC					
		$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ	$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ
IMS	8	0.2529	0.034	0.2056	0.034	0.8566	0.05	0.3181	0.008	**0.0596**	0.009	**0.2881**	0.069
	10	0.3539	0.032	0.2661	0.035	0.9187	0.095	**0.3457**	0.008	**0.083**	0.007	**0.3989**	0.037
	12	0.5792	0.075	0.5095	0.065	1.9259	0.133	**0.3595**	0.013	**0.0826**	0.007	**0.348**	0.072
Monza	8	0.5575	0.022	0.3698	0.017	2.4215	0.117	0.8025	0.008	**0.1991**	0.013	**1.1947**	0.129
	10	0.6651	0.01	0.4417	0.011	2.7088	0.106	0.9045	0.004	**0.1871**	0.012	**1.8055**	0.14
	12	0.6824	0.041	0.4557	0.02	2.7358	0.123	0.9503	0.006	**0.1844**	0.005	**1.1244**	0.108
Silverstone	8	0.6338	0.029	0.4097	0.009	2.3961	0.115	0.7682	0.002	**0.365**	0.002	**1.9831**	0.032
	10	0.793	0.037	0.5195	0.028	3.422	0.277	0.8982	0.008	**0.4041**	0.006	**2.0945**	0.071
	12	0.8179	0.026	0.549	0.017	3.5955	0.189	0.9565	0.006	**0.3972**	0.01	**2.1061**	0.073
Oschersleben	8	1.0396	0.028	**0.4389**	0.029	**1.9817**	0.171	**0.7379**	0.086	0.5186	0.087	3.5435	0.502
	10	1.1044	0.047	**0.4484**	0.027	**2.3037**	0.216	**0.735**	0.043	0.5239	0.051	3.6847	0.555
	12	1.091	0.025	**0.4504**	0.025	**2.1384**	0.081	**0.8039**	0.095	0.5844	0.095	3.9409	0.707

TABLE II: Simulation results for infeasible trajectories

Track	Avg. Speed	Method											
		KFC						NMPC					
		$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ	$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ
IMS	5	**0.1678**	0.01	0.0614	0.012	0.3794	0.158	0.2715	0.011	**0.0305**	0.002	**0.1282**	0.012
	8	**0.3727**	0.043	0.2383	0.058	0.8688	0.07	0.5105	0.049	**0.0697**	0.003	**0.2986**	0.018
	10	**0.4614**	0.044	0.2426	0.062	1.0601	0.429	0.6299	0.054	**0.1115**	0.01	**0.4926**	0.087
	12	0.8229	0.085	0.5981	0.101	2.1172	0.346	**0.7912**	0.039	**0.1278**	0.016	**0.6762**	0.189
Monza	5	0.5947	0.124	0.2933	0.041	2.5874	0.716	**0.4546**	0.007	**0.1454**	0.011	**1.0987**	0.142
	8	1.2623	0.352	0.7826	0.176	4.1392	1.888	**0.9513**	0.031	**0.2896**	0.016	**1.4825**	0.108
	10	**1.8026**	0.079	**1.0903**	0.037	**4.7622**	0.172	2.3195	0.055	1.3282	0.112	8.0575	0.8
	12	**3.1727**	0.154	**1.7966**	0.155	**7.2841**	0.306	5.1239	0.171	3.3295	0.176	18.3838	1.059
Silverstone	5	0.7932	0.073	**0.3003**	0.022	2.2886	0.406	**0.7018**	0.004	0.3322	0.007	**1.6577**	0.098
	6	1.0042	0.086	**0.3771**	0.067	**2.4373**	0.614	**0.9113**	0.02	0.4591	0.01	2.8928	0.168
	7	2.2831	0.492	**0.8071**	0.177	**4.8148**	1.735	**1.6229**	0.058	0.9191	0.031	5.6837	0.289
Oschersleben	5	0.8584	0.05	0.3426	0.01	**1.1911**	0.069	**0.648**	0.031	**0.2706**	0.01	1.191	0.128
	6	**1.2993**	0.07	**0.4627**	0.028	**1.7284**	0.327	1.3127	0.059	0.5087	0.041	2.3236	0.177
	7	**1.8816**	0.141	**0.5602**	0.035	**2.3023**	0.294	2.652	0.066	1.1607	0.019	4.1502	0.253

TABLE III: Real world results for optimal feasible trajectory

Track	Avg. Speed	Method											
		KFC						NMPC					
		$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ	$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ
IMS	5	**0.0929**	0.009	**0.056**	0.004	**0.1664**	0.021	0.5861	0.235	0.1898	0.089	0.8444	0.537
Figure 8	5	**0.114**	0.01	**0.0732**	0.006	**0.2139**	0.059	0.4558	0.072	0.1519	0.036	0.5738	0.31
U - Loop	5	**0.1132**	0.019	**0.0523**	0.002	**0.1874**	0.028	0.4304	0.054	0.1435	0.029	0.6308	0.228

TABLE IV: Real world results for infeasible trajectories

Track	Avg. Speed	Method											
		KFC						NMPC					
		$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ	$\mu\ rmse_t$	σ	$\mu\ rmse_p$	σ	$\mu\ \epsilon$	σ
Loop	1.0	**0.13**	0.05	**0.0636**	0.008	**0.2667**	0.058	0.3541	0.075	0.1387	0.043	0.5833	0.254
	1.5	**0.1897**	0.017	**0.111**	0.009	**0.4815**	0.057	0.4302	0.038	0.1303	0.034	0.4451	0.196
	2.0	**0.447**	0.035	0.2271	0.009	0.7639	0.016	0.5214	0.029	**0.1487**	0.003	**0.4551**	0.033
	3.0	-	-	-	-	-	-	**0.6677**	0.039	**0.2635**	0.042	**0.8918**	0.17
Figure 8	1.0	**0.1688**	0.004	**0.0946**	0.002	**0.2672**	0.011	0.3411	0.029	0.123	0.016	0.3949	0.158
	1.5	**0.3278**	0.006	0.1966	0.006	0.591	0.024	0.4687	0.004	**0.1414**	0.005	**0.3573**	0.03
	2.0	0.7422	0.079	0.2393	0.105	0.7995	0.028	**0.6419**	0.01	**0.1768**	0.002	**0.4827**	0.01
	3.0	-	-	-	-	-	-	**0.9209**	0.019	**0.261**	0.01	**1.0266**	0.017
U - Loop	1.0	0.5023	0.098	0.2812	0.105	1.0001	0.38	**0.4624**	0.142	**0.1645**	0.024	**0.683**	0.23
	1.5	0.7363	0.179	0.1989	0.054	0.8659	0.17	**0.4881**	0.037	**0.156**	0.031	**0.658**	0.127
	2.0	-	-	-	-	-	-	**0.6825**	0.052	**0.2191**	0.018	**0.8731**	0.106
	2.5	-	-	-	-	-	-	**0.7382**	0.045	**0.2541**	0.024	**0.8432**	0.026

TABLE V: Resource utilization by both methods

Method	CPU (%)	Memory (%)
KFC	8.7373	0.7336
NMPC	15.3693	0.8825

results of 5 m/s so they have not been reported in the table. Once again, we see that there is no clear winner. Also, ϵ shows that both controllers deviate beyond the boundary of the track for all speeds except $5m/s$. Additionally, the trend of NMPC consistently having lower position tracking error and max. deviation disappears.

E. Real-world results

Given the small size of the tracks in the real world, the maximum speed tested for infeasible trajectories is 3m/s. Table IV presents the results for individual tracks. As with the simulation, we see that both the controllers perform similarly. However, the performance of KFC degrades at higher speeds to the point that it cannot execute infeasible trajectories for speeds greater than 2 m/s within the ϵ limits whereas NMPC can (in the limited motion capture space, KFC crashes into the track boundary)

In the area of the motion capture setup, there was only one possible optimal trajectory with a $V_{max} = 5m/s$ and $|a_{max}| = 0.5m/s^2$. Lower V_{max} resulted in trajectories nearly identical to infeasible trajectories with speeds of 1,1.5 m/s. For $V_{max} = 5m/s$, the trajectory often maxes out acceleration to the max permissible value. For the feasible trajectory, Table III presents the results for individual tracks. We observe that NMPC's performance is within the range of its performance for infeasible trajectories i.e. it shows no improvements, whereas KFC has improved by a significant margin and showcases extremely low $rmse_t$, $rmse_p$ and ϵ.

F. Resource Usage

Table V shows the resources that each controller used on the testing platform. NMPC uses twice as much CPU as KFC but both methods require the same amount of memory.

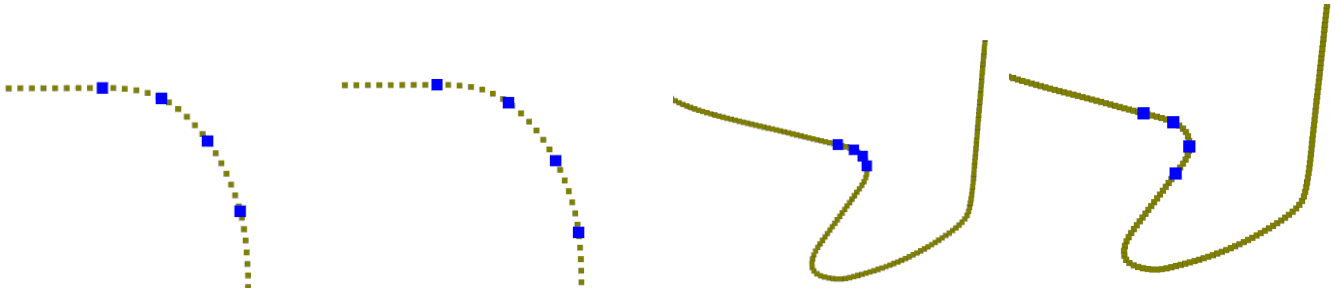
G. Inference

To summarise, given a feasible trajectory that produces smooth accelerations, the performance of KFC is on par with MPC. For other cases, while the data may show that there are instances where NMPC is marginally better, it comes at the cost of 2x more computational overhead. Additionally, the performance of NMPC is highly sensitive to the size of the receding horizon and the discretization step. In following paragraphs we discuss in detail the pros and cons of each and give a detailed analysis of our observations to justify our claim.

All the tests in simulation reveal that KFC keeps up with NMPC or is slightly better. As mentioned earlier, there is no consistent trend in the results that show one method outperforming the other. Any difference in performance is marginal. For example, if we compare the results at $5m/s$ for infeasible trajectories on the Monza track we can see that NMPC does better with a performance delta of 0.1401 m. However, considering that the length of the track is 446m (IV-A) we argue that this marginal difference does not justify the 2x increase in the computational cost.

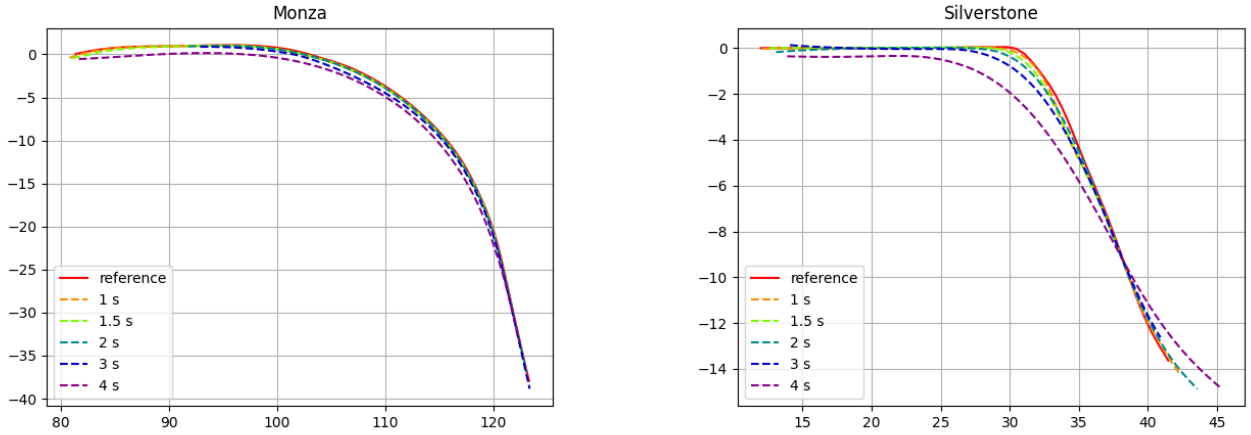
In the experiment, we can see that for infeasible trajectories, at low speeds the results are consistent with simulation. However, when it comes to higher speeds, large values of ϵ prevent KFC from safely executing the trajectories but this is not the case for NMPC. This is due to the fact that accelerations from NMPC are smoother than the accelerations from KFC. The smoother accelerations are a result of the jerk constraint in the formulation as well as due to the fact that the predictions over a finite time horizon make NMPC react to turns earlier resulting in a less overshoot at the turns. In contrast, KFC tracks only one point in the reference and has no constraints on jerk which results in a late deceleration for turns. This causes a non-trivial degree of skid which is large enough to be observed visually without the aid of any sensors. Due to the skid, KFC has low tolerance to high jerk when there is not enough traction.

On feasible trajectories, KFC outperforms NMPC with extremely low errors and ϵ across the board. On the other hand, results for NMPC are similar to its results for infeasible trajectories. This is inconsistent with the simulation tests where we see drastic improvements in its performance. Due to the small size of the track in the real world, there is a minor difference between the states that NMPC optimizes for in the predictive horizon. Fig. 4 shows the difference between



(a) (b) (c) (d)

Fig. 4: Effect of time horizon on real tracks (4a, 4b) vs simulated tracks (4c), 4d for NMPC. Blue markers are the references in the receding horizon



(a) (b)

Fig. 5: Effect of size of the receding horizon on different kinds of turns for NMPC

predictive horizon for feasible trajectories and infeasible trajectories in the real world, 4a (feasible) and 4b (infeasible), and simulation, 4c (feasible) and 4d (infeasible). It can be seen that in the simulation, there is an appreciable difference between the states that NMPC is optimizing for (the size of the time horizon and dt is the same in both cases). The length of the effective arc, in simulation, for a feasible trajectory is $\approx$ 4.05m and $\approx$ 7.2m for infeasible, which is a significant increase. In contrast, in the real world, the effective length of the arcs is $\approx$ 1.8m for feasible and $\approx$ 2.2m for infeasible (the size of the time horizon and dt is the same in both cases as before). From Fig. 4 we can also visually see that in real life NMPC is more or less optimizing for the same states. This results in the performance of NMPC not differing for feasible and infeasible trajectories. Tuning the horizon any further, also does not guarantee optimal performance across the entire track. Fig. 5 shows the effect of size of the receding horizon on different types of turns. It can be seen that for tighter turns 5b a larger horizon results in large deviations from the reference (large horizon results in an undershoot while a small horizon results in under-actuation and sub-optimal control), whereas shallow turns 5a are not as sensitive to the size of horizon. On race-tracks that naturally have a large variation in turns, this is not desirable because it implies that the controller has to be tuned per track. [33] and [30]also identify this problem and the former try to address it with an adaptive horizon based on the curvature of the trajectory. While, effect of the size of the receding horizon in NMPC is desirable when the trajectories are infeasible, it can result in inconsistent performance for different tracks. Additionally, the low computation overhead of KFC frees up the CPU for other critical components in the autonomy pipeline.

V. CONCLUSION & FUTURE WORK

The results of this study shows the competitiveness of differential flatness based controller for tracking high-speed aggressive trajectories for autonomous racing cars. We show that the performance of our proposed controller is on par with the current standard approach for autonomous racing, model predictive control, while having a low computational overhead.Our implementations of both controllers and trajectory generation assumes a perfect bicycle model and considers only kinematics. The positive results of KFC encourage us

to incorporate full state dynamics for both the controllers as well as trajectory generation and reevaluate their performance in the future.

REFERENCES

[1] F. Foundation, "F1tenth homepage," 2022.

[2] D. Robocars, "Diy robocars," 2022.

[3] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *ASME international mechanical engineering congress and exposition*, Citeseer, 1995.

[4] A. Bulsara, A. Raman, S. Kamarajugadda, M. Schmid, and V. N. Krovi, "Obstacle avoidance using model predictive control: An implementation and validation study using scaled vehicles," tech. rep., SAE Technical Paper, 2020.

[5] V. Freire and X. Xu, "Optimal control for kinematic bicycle model with continuous-time safety guarantees: A sequential second-order cone programming approach," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11681–11688, 2022.

[6] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 812–818, 2017.

[7] R. Walambe, N. Agarwal, S. Kale, and V. Joshi, "Optimal trajectory generation for car-type mobile robot using spline interpolationthis work is carried out under the research project grant sanctioned under the wos-a scheme by department of science and technology (dst), govt. of india.," *IFAC-PapersOnLine*, vol. 49, no. 1, pp. 601–606, 2016. 4th IFAC Conference on Advances in Control and Optimization of Dynamical Systems ACODS 2016.

[8] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36, pp. 628–647, July 2014.

[9] E. Mikuláš, M. Gulan, and G. Takács, "Model predictive torque vectoring control for a formula student electric racing car," in *2018 European Control Conference (ECC)*, pp. 581–588, 2018.

[10] D. Kloeser, T. Schoels, T. Sartor, A. Zanelli, G. Prison, and M. Diehl, "NMPC for racing using a singularity-free path-parametric model with obstacle avoidance," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 14324–14329, 2020.

[11] V. S. Babu and M. 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)*, pp. 1614–1620, IEEE, 2020.

[12] M. Fliess, J. Lévine, P. Martin, and P. Rouchon, "On differentially flat nonlinear systems," *IFAC Proceedings Volumes*, vol. 25, no. 13, pp. 159–163, 1992. 2nd IFAC Symposium on Nonlinear Control Systems Design 1992, Bordeaux, France, 24-26 June.

[13] D. Zhao, S. Mishra, and F. Gandhi, "A differential-flatness-based approach for autonomous helicopter shipboard landing," *IEEE/ASME Transactions on Mechatronics*, vol. 27, no. 3, pp. 1557–1569, 2022.

[14] O. Ogunbodede, S. Nandi, and T. Singh, "Periodic Control of Unmanned Aerial Vehicles Based on Differential Flatness," *Journal of Dynamic Systems, Measurement, and Control*, vol. 141, 03 2019. 071003.

[15] B. Hu and S. Mishra, "Time-optimal trajectory generation for landing a quadrotor onto a moving platform," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 585–596, 2019.

[16] C. P. Tang, P. T. Miller, V. N. Krovi, J.-C. Ryu, and S. K. Agrawal, "Differential-flatness-based planning and control of a wheeled mobile manipulator—theory and experiment," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 4, pp. 768–773, 2011.

[17] Z. Zhu, E. Schmerling, and M. Pavone, "A convex optimization approach to smooth trajectories for motion planning with car-like robots," in *2015 54th IEEE Conference on Decision and Control (CDC)*, IEEE, Dec. 2015.

[18] Y. Cong, O. Sawodny, H. Chen, J. Zimmermann, and A. Lutz, "Motion planning for an autonomous vehicle driving on motorways by using flatness properties," in *2010 IEEE International Conference on Control Applications*, IEEE, Sept. 2010.

[19] V. Freire and X. Xu, "Optimal control for kinematic bicycle model with continuous-time safety guarantees: A sequential second-order cone programming approach," *IEEE Robotics and Automation Letters*, vol. 7, pp. 11681–11688, Oct. 2022.

[20] J. hwan Jeon, R. V. Cowlagi, S. C. Peters, S. Karaman, E. Frazzoli, P. Tsiotras, and K. Iagnemma, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *2013 American Control Conference*, IEEE, June 2013.

[21] Z. Han, Y. Wu, T. Li, L. Zhang, L. Pei, L. Xu, C. Li, C. Ma, C. Xu, S. Shen, and F. Gao, "Differential flatness-based trajectory planning for autonomous vehicles," 2022.

[22] 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 Journal of Intelligent Transportation Systems*, vol. 3, pp. 458–488, 2022.

[23] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2018.

[24] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1203–1218, 2020.

[25] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear MPC and differential-flatness-based control for quadrotor agile flight," *IEEE Transactions on Robotics*, vol. 38, pp. 3357–3373, Dec. 2022.

[26] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[27] E. Kreyszig, *Differential Geometry*. Dover Books on Mathematics, Mineola, NY: Dover Publications, June 1991.

[28] E. W. Weisstein, "Curvature."

[29] C. P. Tang, P. T. Miller, V. N. Krovi, J.-C. Ryu, and S. K. Agrawal, "Differential-flatness-based planning and control of a wheeled mobile manipulator—theory and experiment," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 4, pp. 768–773, 2010.

[30] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1094–1099, 2015.

[31] F. Foundation, "F1tenth github page," 2022.

[32] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[33] K. Yang, S. Sukkarieh, and Y. Kang, "Adaptive nonlinear model predictive path tracking control for a fixed-wing unmanned aerial vehicle," in *AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, June 2009.