Comparison of Linear and Nonlinear Model Predictive Control for Vehicle Path Following

Vjekoslav Diklić*, Branimir Novoselnik *,
* Fakultet elektrotehnike i računarstva, Zagreb, Croatia
vjekoslav.diklic@fer.hr branimir.novoselnik@fer.hr

***Abstract*—This technical paper investigates and compares the performance of two model predictive control algorithms employed in the control of a ground vehicle. Both predictive controllers are based on a simplified bicycle model of the vehicle. The main difference is that one controller uses a nonlinear version of the bicycle model and the other one further simplifies it by linearizing it around the operating point. The study centers around evaluating the controllers' effectiveness in a dynamic scenario through the execution of a double-lane change maneuver. By assessing trajectory tracking and computational efficiency, this paper aims to delineate the impact of linearization choices on the controllers' overall performance. Both controllers are extensively tested in realistic simulations on a full non-linear multi-body model of the vehicle sourced from the CommonRoad Python library. The results of these simulations provide insights into the trade-offs between model accuracy and computational complexity.**

***Keywords*—nonlinear model predictive control, linear model predictive control, ground vehicle, bicycle model, multi-body model**

I. INTRODUCTION

The challenge of controlling a road vehicle has grown in tandem with automotive innovation, initially stemming from the complex interplay of mechanical components and expanding with technological advancements. The quest for effective vehicle control encompasses a myriad of factors, including the need to optimize propulsion systems, ensure stability and manoeuvrability, and implement sophisticated control algorithms capable of adapting to diverse driving conditions [1]. As technology has advanced, so too has the challenge, with the integration of autonomous features, machine learning, and sensor fusion presenting a frontier where the pursuit of safer, more efficient, and increasingly automated road transportation continues to captivate researchers, engineers, and enthusiasts alike.

An appropriate maneuver that can be used to test the performance of a vehicle is the so-called double lane change *(DLC)* proposed by F. Borrelli et al. [2]. This maneuver consists of two distinctive aggressive steering actions: one lane change left and then a double lane change right. Performing such a maneuver is a comprehensive test of the vehicle's stability and its behaviour on various road surfaces. Many authors have used the DLC maneuver to evaluate their vehicle controllers. J.Z.Y. Lu et al. [3] used DLC for testing the proposed Blend Path Curvature Control [3]. A. Domina et al. have evaluated the proposed linear time-varying Model Predictive Control *(MPC)* algorithm [4]. F. Borrelli et al. [2] have used the nonlinear MPC approach for the DLC maneuver.

MPC has emerged as a powerful methodology in control systems and optimization, particularly for its effectiveness in managing complex and dynamic systems [5]. It is characterized by its ability to predict the future behaviour of a system and optimize control inputs based on these predictions. The adaptability and precision of MPC make it ideal for vehicular control, where it addresses the intricacies of modern vehicle dynamics [6].

This technical paper presents a performance comparison between two MPC approaches: one based on a bicycle model and the other based on a linearised bicycle model. The paper is organized as follows: Section II describes the main problem addressed in this article further. Section III presents both linear and nonlinear implementations of MPC for the given problem. Evaluation and test results are available in Section IV, while the conclusion is provided in Section V.

II. PROBLEM DESCRIPTION

The solution to the vehicle path-following problem can be decomposed into three sub-problems: vehicle modelling and simulation, reference path definition, and controller selection. For the vehicle model, a multi-body vehicle model from the Python-based Common Road Vehicle Library [7] was chosen. This model accounts for the vertical load on all four wheels due to roll, pitch, and yaw, as well as their individual spin and slip, incorporating a nonlinear tire dynamics model. For tire dynamics, this model utilizes the PAC2002 Magic Formula tire model [8]. Such advanced vehicle and tire models are commonly employed in motion planning for road vehicles [9].

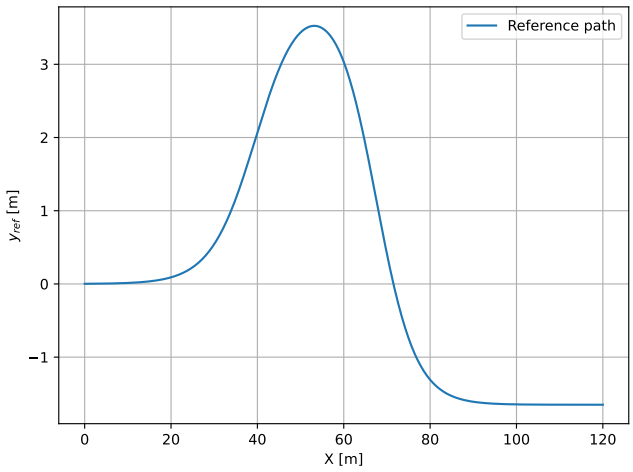
The reference path for the vehicle's movement is defined by the earlier mentioned DLC maneuver,shown on fig. 1. DLC is characterized by the following equations:

$$y_{\text{ref}} = \frac{d_{y_1}}{2}\left(1+\tanh(z_1)\right) - \frac{d_{y_2}}{2}\left(1+\tanh(z_2)\right) \quad (1)$$

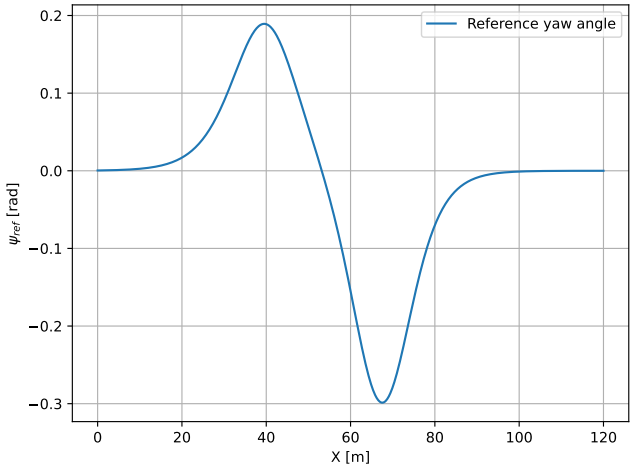
$$\psi_{\text{ref}} = \arctan\left(\frac{d_{y_1}\cdot 1.2}{d_{x_1}\cdot\cosh(z_1)^2} - \frac{d_{y_2}\cdot 1.2}{d_{x_2}\cdot\cosh(z_2)^2}\right) \quad (2)$$

$$z_1 = \frac{shape}{d_{x_1}}\left(X - X_{s_1}\right) - \frac{shape}{2} \quad (3)$$

$$z_2 = \frac{shape}{d_{x_2}}\left(X - X_{s_2}\right) - \frac{shape}{2} \quad (4)$$



(a) DLC position reference.



(b) DLC yaw angle reference.

Fig. 1: DLC reference values.

eqs. (1) to (4) yield the reference lateral vehicle position y_{ref} and the reference vehicle yaw angle ψ_{ref}. The $shape = 2.4$ parameter influences the curvature of the lane change, while $d_{x_1} = 25$ meters and $d_{x_2} = 21.95$ meters denote the longitudinal distances of the first and second lane changes, respectively. Lateral shifts are specified by $d_{y_1} = 4.05$ meters for the first and $d_{y_2} = 5.7$ meters for the second lane change. The starting points for these maneuvers are marked at $X_{s_1} = 27.19$ meters and $X_{s_2} = 56.46$ meters along the path.

As mentioned earlier, the controller for path following will be implemented in the form of MPC algorithm. Such a controller requires a mathematical model of the system it is controlling for behaviour predictions. There will be two different mathematical models used for the MPC controller: a linearised variant and a nonlinear variant. The implementation of the MPC controller and the mathematical models will be further detailed in Section III.

The simulation loop comprises three primary components: a reference path, an MPC controller, and a vehicle model. Each simulation step begins with the current vehicle state X_k, from which the reference values for lateral position Y_{ref} and yaw angle ψ_{ref} are derived using DLC equations (eq. (1)) - (eq. (4)) form state variable p_x for vehicle global position x coordinate. The current vehicle state X_k, together with the reference values Y_{ref} and ψ_{ref}, is then input into the MPC, which calculates the optimal control values u_k. Subsequently, u_k, along with the current vehicle state X_k, is used to compute the vehicle's next state.

III. Model Predictive Controllers

Model Predictive Control has emerged as a powerful paradigm for dynamic systems, offering a systematic framework for trajectory optimization and control. The essence of MPC lies in forecasting the future state of a dynamic system through the simulation of its mathematical model. The model utilized for predicting the future behavior of the system is often characterized by simplicity, featuring fewer state variables than the real system. This approach is driven by two primary constrains. Firstly, it is often impractical and/or impossible to model all segments of the system due to inherent complexities. Secondly, the adoption of a simplified model is motivated by the computational constraints associated with the processing time required for the calculation of the subsequent state. MPC calculates optimal control values by minimizing the following criterion function:

$$\mathbf{J} = \sum_{i=0}^{N-1} \left(\mathbf{x}_i^T \mathbf{Q} \mathbf{x}_i + \mathbf{u}_i^T \mathbf{R} \mathbf{u}_i \right) + \mathbf{x}_N^T \mathbf{S_N} \mathbf{x}_N \qquad (5)$$

This equation represents the cost function for the predictive controller in a quadratic form. Summation occurs over the prediction horizon N, where $\mathbf{x}_i$ and $\mathbf{u}_i$ denote the state and input vectors at time step i. The matrices $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S_N}$ are the weighting matrices in the cost function. The full nonlinear mathematical model, which is used to describe the real system, comprises 29 state variables and two control inputs:

- $\dot{\delta}$: the rate of change of the steering angle.
- a: acceleration.

The state variables are grouped into four categories: vehicle body, front axle, rear axle and wheels [7]. This high-fidelity model is used in simulations to test and validate the developed MPC algorithm. However, the MPC algorithm itself is based on a simplified mathematical model comprising only five state variables $(p_x, p_y, v_x, \psi, \delta)$, the so-called bicycle model [10] (see fig. 2). The variables p_x and p_y denote the current position of the vehicle within the global coordinate system, v_x represents the current longitudinal speed of the vehicle, ψ is the yaw angle of the vehicle's direction, and δ corresponds to the steering value of the front axle.

A. Vehicle bicycle model

The vehicle bicycle model represents a simplified of a four-wheeled road vehicle. Its name implies a two-wheeled configuration, where the front wheel serves solely as a steering component, while the rear wheel assumes a

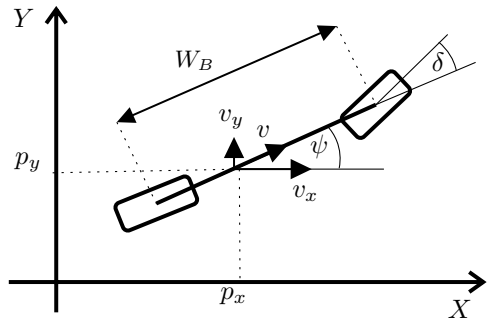


Fig. 2: Bicycle model parameters.

passive role. Propulsion and braking forces are modelled to act at the vehicle's centre of mass. The global position (p_x, p_y) of the vehicle in a bicycle model can be defined by the following equation:

$$\begin{aligned} \frac{dp_x}{dt} &= v\cos(\psi), \\ \frac{dp_y}{dt} &= v\sin(\psi). \end{aligned} \tag{6}$$

Velocity can be described as a function of acceleration through time.

$$\frac{dv}{dt} = a. \tag{7}$$

The effects of steering and yaw angle are modeled using the following equation:

$$\begin{aligned} \frac{d\psi}{dt} &= \frac{v}{W_B}\tan(\delta), \\ \frac{d\delta}{dt} &= \dot{\delta}. \end{aligned} \tag{8}$$

Now, it is possible to express everything as a state-space system.

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{y} &= \mathbf{h}(\mathbf{x}, \mathbf{u}) \end{aligned} \tag{9}$$

where $\mathbf{x}$ is a state vector and $\mathbf{u}$ is the input vector.

$$\mathbf{x} = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \delta \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \dot{\delta} \\ a \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{10}$$

The complete state-space description of the bicycle model is given as follows:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \end{bmatrix} = \begin{bmatrix} x_3\cos(x_4) \\ x_3\sin(x_4) \\ u_2 \\ \frac{x_3}{W_B}\tan(x_5) \\ u_1 \end{bmatrix}. \tag{11}$$

The linearisation of this model will be further covered in section III-C.

B. Model discretization

To utilize the bicycle model from section III-A as prediction model for MPC, it is necessary to convert it into a discrete version. The discretization process for a continuous-time system described by the state-space equations in eq. (9) can be achieved using the Euler method. The Euler method is a straightforward numerical integration technique used for approximating the solution of ordinary differential equations *ODE* or, in this context, for discretizing continuous-time systems. The basic idea is to approximate the derivative by a finite difference and use this approximation to update the state variables over small time intervals. The discrete-time state update can be achived eq. (12):

$$\mathbf{x}^+ = \mathbf{x} + \Delta t \cdot \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{12}$$

In this context, $\mathbf{x}^+$ refers to the state at time step $k+1$, $\mathbf{x}$ represents the state at time step k, Δt is the duration of each time step, $\mathbf{f}$ describes the system dynamicsand $\mathbf{u}_k$ is the input at time step k. By substituting 11 into eq. (12), the discrete nonlinear bicycle model can be expressed as follows:

$$\mathbf{x}^+ = \begin{bmatrix} x_1 + \Delta t \cdot x_3\cos(x_4) \\ x_2 + \Delta t \cdot x_3\sin(x_4) \\ x_3 + \Delta t \cdot u_2 \\ x_4 + \Delta t \cdot \frac{x_3}{W_B}\tan(x_5) \\ x_5 + \Delta t \cdot u_1 \end{bmatrix} \tag{13}$$

C. Linearisation

Linearisation is a method that approximates nonlinear functions with linear counterparts around a certain operating point. As the distance from the operating point increases, the approximation error also rises. To mitigate the impact of this issue, the common approach is to update the operating point more frequently. Linearisation is commonly performed by utilizing the first-order Taylor series expansion around a operating point for a given nonlinear function. First order Taylor expression is given in eq. (14).

$$f(x) \approx f(\bar{x}) + f'(\bar{x}) \cdot (x - \bar{x}) + \dots \tag{14}$$

Here, $f(\bar{x})$ corresponds to the value of the function at the point $\bar{x}$, while $f'(\bar{x})$ represents the first derivative of the function evaluated at $\bar{x}$. For a multivariate function, such as the bicycle model given in eq. (13), denoted as $\mathbf{f}(\mathbf{x})$, where $\mathbf{x}$ is a vector, the Taylor series expansion around a point $\bar{\mathbf{x}}$ is given by eq. (15).

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\bar{\mathbf{x}}) + \nabla\mathbf{f}(\bar{\mathbf{x}}) \cdot (\mathbf{x} - \bar{\mathbf{x}}) + \dots \tag{15}$$

In this context, $f(\bar{\mathbf{x}})$ signifies the value of the function at the point $\bar{\mathbf{x}}$, while $\nabla f(\bar{\mathbf{x}})$ denotes the gradient of f evaluated at operating point defined as follows:

$$\overline{\mathbf{x}} = \begin{bmatrix} \overline{x_1} & \overline{x_2} & \overline{x_3} & \overline{x_4} & \overline{x_5} \end{bmatrix}^T \tag{16}$$

substituting $\mathbf{x}^+$ from eq. (13) into eq. (15) as $\mathbf{f}(\mathbf{x})$, gives linearised system:

$$\mathbf{x}^+ \approx \begin{bmatrix} x_1 + \Delta t \cdot \cos(\overline{x_4})x_3 - \Delta t \cdot \overline{x_3}\sin(\overline{x_4})x_4 \\ x_2 + \Delta t \cdot \sin(\overline{x_4})x_3 + \Delta t \cdot \overline{x_3}\cos(\overline{x_4})x_4 \\ x_3 + \Delta t \cdot u_2 \\ x_4 + \Delta t \cdot \frac{1}{W_B}(\tan(\overline{x_5})x_3 + \overline{x_3}\sec^2(\overline{x_5})x_5) \\ x_5 + \Delta t \cdot u_1 \end{bmatrix} \tag{17}$$

Where elements $x_1, x_2, ..., x_5$ represent small offsets from operation point. Now matrix $\mathbf{A_L}$ for linearised version of vehicle bicycle model matrices can be written as follows:

$$\mathbf{A_L} = \begin{bmatrix} 1 & 0 & \Delta t \cos(\overline{x_4}) & -\Delta t \cdot \overline{x_3} \sin(\overline{x_4}) & 0 \\ 0 & 1 & \Delta t \sin(\overline{x_4}) & \Delta t \cdot \overline{x_3} \cos(\overline{x_4}) & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \Delta t \frac{\tan(\overline{x_5})}{W_B} & 1 & \Delta t \frac{\overline{x_3} \sec^2(\overline{x_5})}{W_B} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

And matrix $\mathbf{B_L}$ as follows:

$$\mathbf{B_L} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & \Delta t \\ 0 & 0 \\ \Delta t & 0 \end{bmatrix} \tag{19}$$

And now system can be written as

$$\dot{\mathbf{x}} = \mathbf{A_L}\mathbf{x} + \mathbf{B_L}\mathbf{u} \tag{20}$$

IV. Evaluation and Results

The evaluation simulations and MPC optimization problem solving were conducted on a personal computer equipped with an Intel Core i7-8850H CPU, boasting a base frequency clock of 2.60 GHz with up to 400 GFLOPS, and 64 GB of system memory. The *CasADi* symbolic framework for Python [11], was employed for defining optimization problems, while for solving optimization problems the Interior Point Optimizer *Ipopt* [12] was utilized. Model simulation was performed using the ordinary differential equation integration function (*odeint*) from the Scipy Python library [13], with a relative tolerance $r_{tol} = 10^{-3}$ and an absolute tolerance $a_{tol} = 10^{-6}$. The vehicle parameters utilized in the evaluation were sourced from the Common Road Vehicle Library. Specifically, for the evaluation, the vehicle with identification $id = 2$ was selected, and its parameters are presented in table I. Tire dynamics parameters were used for dry asphalt. The evaluation involved the application of both linear and nonlinear MPC with a sampling time $T_{\text{sample}} = 0.025$ seconds. Both linear and nonlinear MPC had set limitation for maximum input values:

$$\begin{aligned} -11.5\ \text{m/s}^2 \leq a \leq 11.5\ \text{m/s}^2 \\ -0.4\ \text{rad} \leq \dot{\delta} \leq 0.4\ \text{rad} \end{aligned} \tag{21}$$

The required parameter for the wheelbase (W_B) in the bicycle model was set to 2.5 meters. Tests were conducted at four different initial vehicle velocities ($5, 10, 15, 17$ m/s) and utilized three sizes of prediction horizons (h_p) $(2, 7, 10)$, while the control horizon (h_u) ranged from one up to the value of the prediction horizon. This setup resulted in a total of 76 simulations for each velocity level, and when multiplied by the number of test velocities (4), it equated to 152 simulations. Therefore, in total, 204 simulations were conducted for linear and nonlinear MPC. It is noteworthy that both the linear and nonlinear

TABLE I: Vehicle parameters

Property	Value
Vehicle Mass	1000 kg
Vehicle Length	4.508 m
Vehicle Width	1.610 m
Wheel Base	2.578 m
Max. steering angle	1.066 rad
Max. steering velocity	0.4 rad/s
Max. acceleration	$11.5\,m/s^2$
Max. velocity	$50.8\,m/s$

MPC formulations utilized the same $\mathbf{Q}, \mathbf{R}, \mathbf{S_N}$ weighting matrices:

$$\mathbf{Q} = \begin{bmatrix} 20 & 0 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 200 & 0 \\ 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{22}$$

$$\mathbf{S_N} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 1000 & 0 \\ 0 & 0 & 0 & 0 & 0.1 \end{bmatrix}. \tag{23}$$

These parameters were obtained by manual adjustment in a series of simulations with the aim of optimizing the matching of the referent trajectory. The bicycle model operating point for linear MPC was update with last know state each step. The results presented in table II originate from the best-performing MPC corresponding to a given velocity. This determination is grounded in selecting the configuration with the smallest mean squared error ϵ defined in eq. (24) between the vehicle's positional value and the reference path value. Selection process considers every combination of initial vehicle velocity and each of prediction horizon h_p sizes. Table II shows only control horizons h_u that are giving the best results for given combination of velocity and prediction horizon.

$$\epsilon = \frac{1}{2n} \sum_{i=1}^{n} (p_{x,i} - \overline{p_{x,i}})^2 + (p_{y,i} - \overline{p_{y,i}})^2 \tag{24}$$

The value marked with t_c, representing the average time necessary to solve the optimization problem in each step, excludes the time to shift the operating point in the case of a linearised model. Time t_m presents maximum time needed for solving optimization problem. Table values are highlighted in bold font for the parameter t_c to signify the MPC configuration with the smallest optimization solving time, whereas ϵ indicates the configuration with the smallest error. The empty field are there to indicate that MPC with such configuration of h_p and h_u is not usable for given vehicle velocity. The dependencies of error ϵ combined with prediction horizon size and vehicle velocity, are illustrated in Fig. 3. figs. 4 to 7 illustrate simulation plots for the MPC configuration that exhibited the smallest mean squared error. These figures clearly show advantages of the linearized model in computation time, as well as its disadvantage in performance, best demonstrated

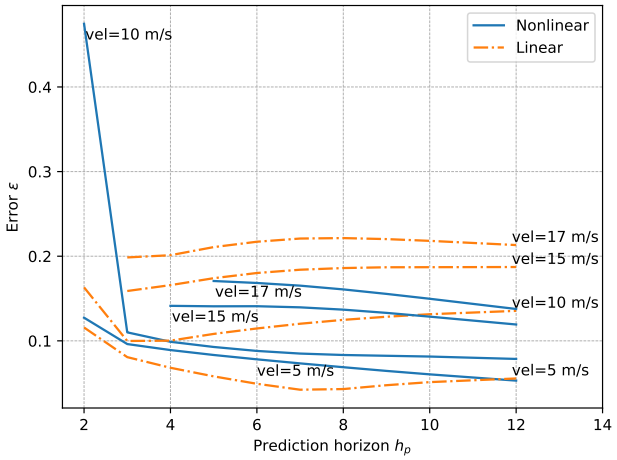


Fig. 3: Path fallowing error compared the length of prediction horizon.

TABLE II: Evaluation results.

Vel. [m/s]	Value	Prediction horizont h_p = 2 Lin.	2 Non. Lin.	7 Lin.	7 Non. Lin.	10 Lin.	10 Non. Lin.
5	h_u	2	2	3	1	10	1
	$t_c[s]$	**0,00763**	**0,00846**	0,01259	0,01468	0,01677	0,01944
	$t_m[s]$	0,01464	0,01562	0,01855	0,01953	0,03221	0,03709
	ϵ	0,13698	0,17665	**0,05435**	0,07416	0,06168	**0,06486**
10	h_u	2	2	7	7	10	2
	$t_c[s]$	**0,00798**	**0,00883**	0,01342	0,01541	0,01678	0,01943
	$t_m[s]$	0,01272	0,01464	0,02342	0,02827	0,02635	0,03318
	ϵ	0,15954	0,33594	**0,11869**	0,09495	0,12384	**0,09036**
15	h_u			7	7	10	10
	$t_c[s]$			**0,01347**	**0,01594**	0,01712	0,02084
	$t_m[s]$			0,02147	0,02636	0,03022	0,03511
	ϵ			0,18050	0,15090	**0,17900**	**0,14008**
17	h_u			7	7	10	10
	$t_c[s]$			**0,01367**	**0,01595**	0,01721	0,02059
	$t_m[s]$			0,02440	0,02245	0,02831	0,03513
	ϵ			0,21278	0,17571	**0,20699**	**0,16144**

in the yaw rate and path following plots. Additionally, the aforementioned differences are also present in the corresponding values, which are bolded in table II.

V. Conclusion

The results shown in table table II indicate that the sizes of the prediction horizon h_p and control horizon h_u are crucial parameters for vehicle control. It is important to emphasize that a prediction horizon that is too short limits the top velocity at which the vehicle can be successfully controlled with the given MPC. Increasing the prediction horizon comes with the advantage of smaller errors in path following, but it also requires more time to solve the optimization problem in each step. The time required to solve the optimization problem is a crucial parameter, especially for controlling real-time systems like vehicles. On average, the computation of nonlinear MPC was $74,89\%$ slower than that of linear MPC. Additionally, the computation

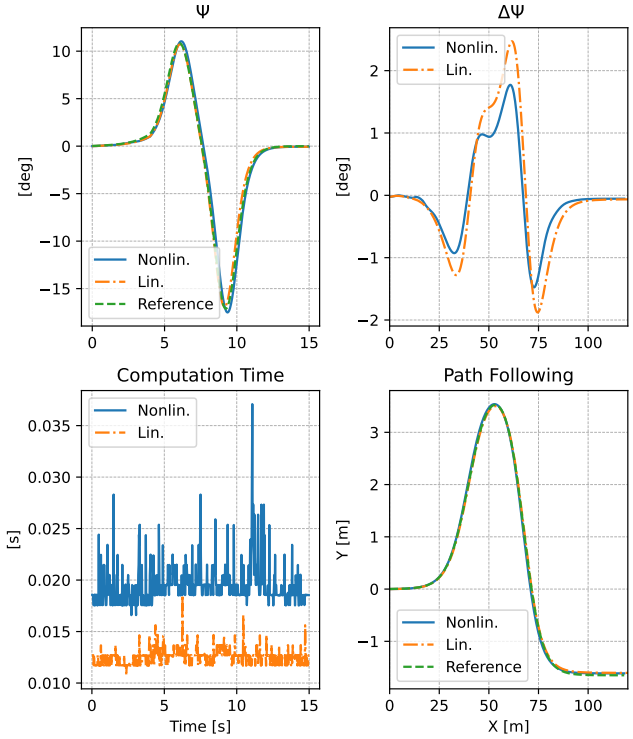


Fig. 4: Results: $v = 5$ m/s; Lin.: h_p = 7, h_u = 3; Nonlin.: h_p = 10, h_u = 1. Yaw angle (Ψ) and Yaw angle error ($\Delta\Psi$), Computation Time and vehicle position (X and Y respectively).

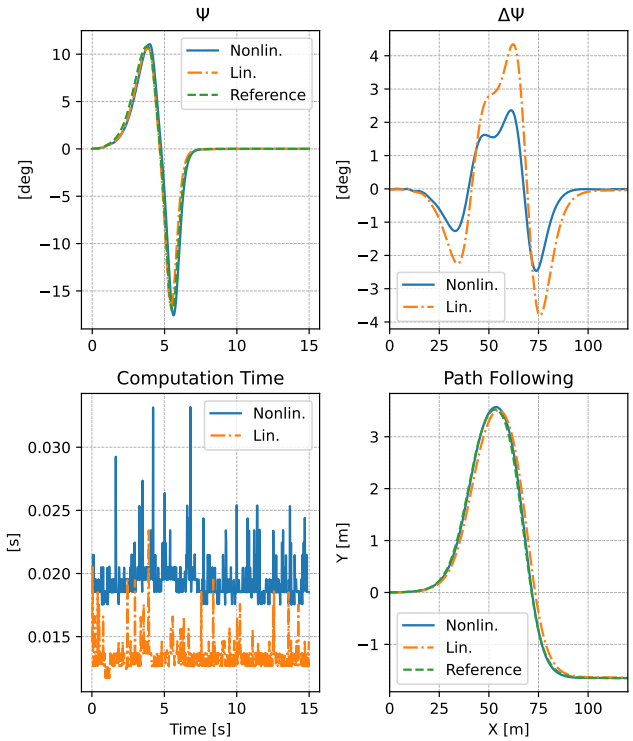


Fig. 5: Results: $v = 10$ m/s; Lin.: h_p = 7, h_u = 7; Nonlin.: h_p = 10, h_u = 2. Yaw angle (Ψ) and Yaw angle error ($\Delta\Psi$), Computation Time and vehicle position (X and Y respectively).

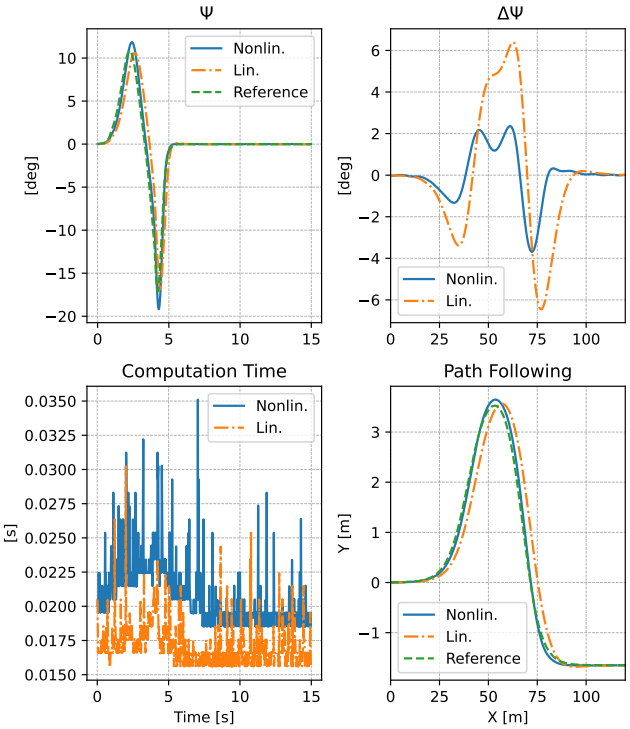


Fig. 6: Results: $v = 15$ m/s; Lin.: h_p = 7, h_u = 7; Nonlin.: h_p = 10, h_u = 10. Yaw angle (Ψ) and Yaw angle error ($\Delta\Psi$), Computation Time and vehicle position (X and Y respectively).

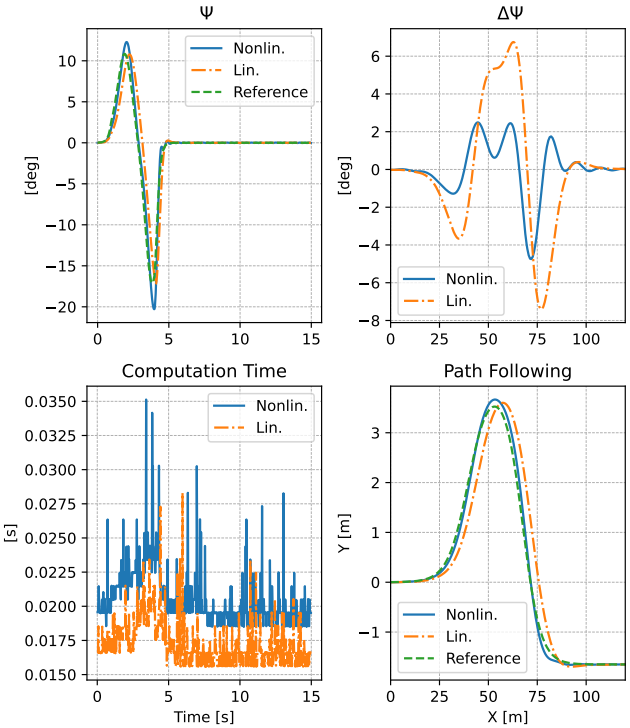


Fig. 7: Results: $v = 17$ m/s; Lin.: h_p = 10, h_u = 10; Nonlin.: h_p = 10, h_u = 10. Yaw angle (Ψ) and Yaw angle error ($\Delta\Psi$), Computation Time and vehicle position (X and Y respectively).

of linear MPC was faster than that of nonlinear MPC in $98,25\%$ of computation instances through simulations. In the fastest instance, linear MPC is almost 4 times faster than nonlinear MPC, with absolute time difference of around 24 ms. The MPC weighting matrices $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S_N}$ can be optimized further. The operation point shift could be updated less frequently to increase the average time necessary to solve the optimization process faster for MPC with the linearised bicycle model. In future work, it should be considered to automatically tune the length of prediction horizon and values of weighting matrices $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{S_N}$ according to the vehicle state, specifically velocity.

References

[1] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, p. 789–814, Jun. 2000. [Online]. Available: http://dx.doi.org/10.1016/S0005-1098(99)00214-9

[2] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. H. , "Mpc-based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Autonomous Systems*, vol. 3, no. 2-4, pp. 265–291, 2005.

[3] J. Z. Y. Lu, A. Abualfellat, and R. Zarringhalam, "Achieving automated vehicle path following with blend path curvature control," in *2022 American Control Conference (ACC)*, 2022, pp. 3158–3163.

[4] A. Domina and V. Tihanyi, "Ltv-mpc approach for automated vehicle path following at the limit of handling," *Sensors*, vol. 22, no. 15, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/15/5807

[5] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model predictive heuristic control," *Automatica*, vol. 14, no. 5, p. 413–428, Sep. 1978. [Online]. Available: http://dx.doi.org/10.1016/0005-1098(78)90001-8

[6] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, p. 33–55, Mar. 2016. [Online]. Available: http://dx.doi.org/10.1109/TIV.2016.2578706

[7] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017.

[8] H. B. Pacejka, *Tire and Vehicle Dynamics*, 3rd ed. Oxford, England: Butterworth-Heinemann, apr 2012.

[9] E. Bertolazzi, F. Biral, and M. D. Lio, "real-time motion planning for multibody systems: Real life application examples," *Multibody System Dynamics*, vol. 17, no. 2–3, p. 119–139, Feb. 2007. [Online]. Available: http://dx.doi.org/10.1007/s11044-007-9037-7

[10] P. Polack, F. Altche, B. d'Andrea 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)*. IEEE, Jun. 2017. [Online]. Available: http://dx.doi.org/10.1109/IVS.2017.7995816

[11] 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.

[12] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, p. 25–57, Apr. 2005. [Online]. Available: http://dx.doi.org/10.1007/s10107-004-0559-y

[13] 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.