ECE/CS/ME 539 Introduction to Artificial Neural Networks

Project Report

NMPC-Net: An efficient alternative to Nonlinear Model Predictive Control

Team #: 5
Team members:
Ian Ruh, AMEP, Undergraduate
Victor Freire, Mechanical Engineering, Graduate
Date of submission: 05/06/2022

ABSTRACT

We trained a neural network (NN) to control autonomous vehicles. The network was trained to imitate a nonlinear model predictive controller (NMPC) to track setpoint references with the kinematic bicycle model. The resulting controller is called NMPC-Net. The motivation for this project stems from the difficulty of solving NMPC with existing methods. In contrast, evaluating NMPC-Net is computationally cheap. We tested NMPC-Net in simulation and compared its tracking performance with that of the original NMPC it was trained from.

The project GitHub repository is [1].

1 Introduction

In this project, we trained a neural network-based controller to achieve real-time, high-performance setpoint tracking for the kinematic bicycle model. The training data was generated by a nonlinear model predictive controller (NMPC) solved with IPOPT [2] and Pyomo [3, 4].

In the field of optimal control, MPC is usually regarded as a standard of performance. When this control approach is applied to linear systems, we can achieve excellent performance with negligible computational burden. However, MPC applied to nonlinear systems (abbreviated NMPC) requires solving a nonconvex optimization problem at each sampling time. Nonconvex optimization is NP-hard [5] and no polynomial-time solver exists. Unfortunately, most systems worth studying are nonlinear and so is the kinematic bicycle model. To overcome this, we trained a neural network to approximate the solution of the nonconvex optimization problem. Because neural network evaluations happen fast, we can use the trained neural network as the controller for the kinematic bicycle model in place of the NMPC, thus achieving real-time control.

Nonlinear MPC is a classical problem in optimal control and has been studied specifically for the bicycle kinematic model in many works such as [6, 7]. Neural networks have been used to approximate MPC in the literature [8, 9, 10, 11]. Specifically in [8, 11], they used a vehicle model very similar to the kinematic bicycle model. These works served as a starting point for the design of our NMPC-Net.

1.1 Kinematic Bicycle Model

The kinematic bicycle model is simple but captures the nonholonomic constraint present in most wheeled vehicles, and it can be expressed as [7]:

$$\dot{\mathbf{x}}(t) = f\big(\mathbf{x}(t), \mathbf{u}(t)\big) = \begin{bmatrix} v(t)\cos\psi(t) \\ v(t)\sin\psi(t) \\ \dot{v}(t) \\ v(t)\tan\gamma(t)/L \end{bmatrix}, \tag{1}$$

where $\mathbf{x} = (x, y, v, \psi)^T$ and $\mathbf{u} = (\dot{v}, \gamma)^T$ are the state and input vectors, respectively, (x, y) is the position of the rear wheel, v is the magnitude of the velocity vector, ψ is the heading with respect to the inertial frame's x-axis and γ is the steering angle (see Figure 1).

In practice, we can only apply controls $\mathbf{u}$ at discrete times. Therefore, for controller design, we discretize the kinematic bicycle model (1) with a simple forward-Euler integrating scheme to obtain the discretized kinematic bicycle model:

$$\mathbf{x}_{k+1} = f_d(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k + \Delta t \begin{bmatrix} v_k\cos\psi_k \\ v_k\sin\psi_k \\ \dot{v}_k \\ v_k\tan\gamma_k/L \end{bmatrix}, \tag{2}$$

where $\Delta t = 0.1$ seconds is the sampling time used for the discretization.

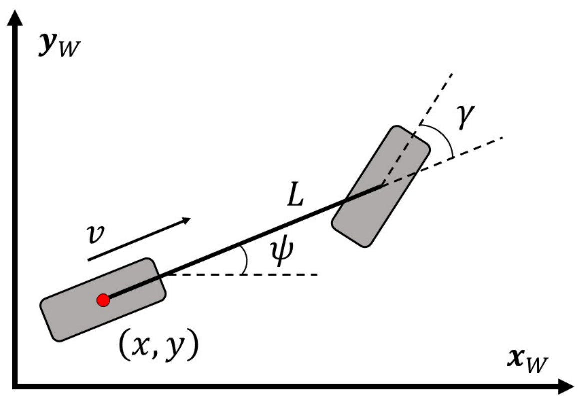


Figure 1: Bicycle kinematic model and world (inertial) coordinate frame. In all examples, we use $L = 2.675$ meters which is the wheelbase length of a 2022 Chevy Bolt EUV.

1.2 Controllers

A *controller* for the kinematic bicycle model is a function $\zeta : \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}^2$ which maps any state $\mathbf{s} \in \mathbb{R}^4$ and a given constant goal state $\mathbf{s}_g \in \mathbb{R}^4$ to an appropriate input $\mathbf{u}$. That is, $\mathbf{u} = \zeta\big(\mathbf{s}, \mathbf{s}_g\big)$. The controller ζ should be designed such that, given an initial state $\mathbf{s}_0 \in \mathbb{R}^4$, the system sequence $\mathbf{x}_k$ satisfies:

$$\begin{aligned} \mathbf{x}_0 &= \mathbf{s}_0, \\ \mathbf{x}_{k+1} &= f_d\big(\mathbf{x}_k, \zeta(\mathbf{x}_k, \mathbf{s}_g)\big), \quad k = 0, 1, 2, \ldots \\ \mathbf{x}_k &\to \mathbf{s}_g \text{ as } k \text{ increases.} \end{aligned}$$

In this work, we consider the following controllers:

- Nonlinear model predictive controller ζ_{MPC} described in Section 1.3.
- Neural network controller ζ_{NN} described in Section 2.

1.3 Nonlinear Model Predictive Control

The controller ζ_{MPC} requires solving an optimization problem each time it is called. For Nonlinear MPC, that optimization problem is usually nonconvex and difficult to solve. This is the main drawback of this controller. The optimization problem is formulated as follows [12]:

$$\begin{aligned} \text{minimize} \quad & \tilde{\mathbf{x}}_N^T Q_f \tilde{\mathbf{x}}_N + \sum_{i=0}^{N-1} \big(\tilde{\mathbf{x}}_i^T Q \tilde{\mathbf{x}}_i + \mathbf{u}_i^T R \mathbf{u}_i\big), \qquad \text{(NMPC)} \\ \text{subject to} \quad & \mathbf{x}_{i+1} = f_d(\mathbf{x}_i, \mathbf{u}_i), \quad k = 0, \ldots, N-1, \\ & \tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{s}_g, \quad i = 0, \ldots, N, \\ & \underline{\mathbf{u}} \le \mathbf{u}_j \le \overline{\mathbf{u}}, \quad j = 0, \ldots, N-1, \\ & 0 \le v_i \le \overline{v}, \quad i = 1, \ldots, N, \\ & \mathbf{x}_0 = \mathbf{s}_k, \end{aligned}$$

over variables $\mathbf{x}_i \in \mathbb{R}^4,\ i = 0, \dots N,\ \mathbf{u}_j \in \mathbb{R}^2,\ j = 0, \dots, N-1$ and given optimization parameters: Goal state $\mathbf{s}_g \in \mathbb{R}^4$, current system state $\mathbf{s}_k \in \mathbb{R}^4$, terminal state weight matrix $Q_f \in \mathbb{S}^4_+$, path state weight matrix $Q \in \mathbb{S}^4_+$, horizon $N = 100$, path input weight matrix $R \in \mathbb{S}^2_{++}$, input bounds $\underline{\mathbf{u}} = (-5, -\pi/4)^T$, $\overline{\mathbf{u}} = (1, \pi/4)^T$ and maximum speed $\overline{v} = 32$ m/s. We used the following weight matrices $Q = \mathrm{diag}(2, 2, 1, 100)$, $R = 100\mathrm{diag}(1, 2)$ and $Q_f = 100Q$.

Let $\mathbf{x}_i$ and $\mathbf{u}_j$ be the solution of (NMPC). Then, we can define ζ_{MPC} as follows:

$$\zeta_{\text{MPC}}(\mathbf{s}_k, \mathbf{s}_g) = \mathbf{u}_0. \tag{3}$$

In other words, only the first input $\mathbf{u}_0$ from the sequence $\mathbf{u}_j,\ j = 0, \dots, N$ is actually applied to the system. The rest of the inputs and all $\mathbf{x}_i$ obtained from the solution of (NMPC) are discarded.

2 Method

In this section, we describe in detail the design (and training) of the neural network controller ζ_{NN}.

The problem (NMPC) is solved at a specified frequency as the system evolves over time. Therefore, the $\mathbf{x}_0$ optimization parameter is updated for each instance of (NMPC) to match the current state of the real system. On the other hand, the $\mathbf{x}_f$ optimization parameter remains fixed through the entire simulation/experiment.

2.1 Dataset

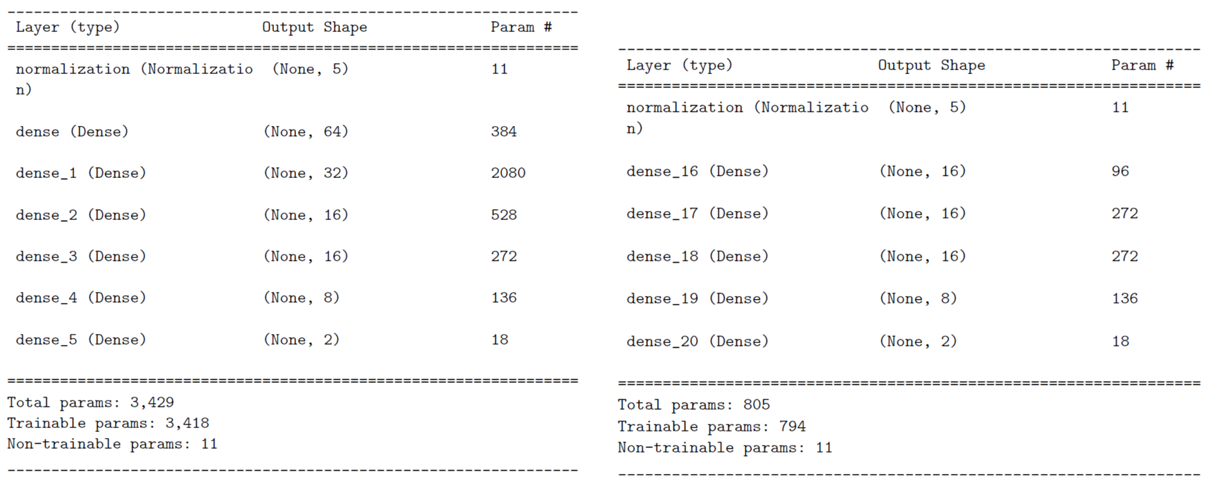
The dataset consists of pairs of current system states $\mathbf{s}_k$ and goal states $\mathbf{s}_f$ as features. That is, our feature vectors are $(\mathbf{s}_k, \mathbf{s}_f) \in \mathbb{R}^4 \times \mathbb{R}^4$. For each feature vector, we solve the corresponding instance of (NMPC) and assign a label according to the controller $\zeta_{\text{MPC}}(\mathbf{s}_k, \mathbf{s}_g)$. That is, we apply the NMPC controller ζ_{MPC} to each feature to calculate its corresponding label. The features are generated pseudo-randomly according to the following procedure: We begin by choosing randomly from uniform distributions a radius $r \in [30, 100]$, direction angle $\theta \in [-\pi/4, \pi/4]$, heading angle $\psi \in [-\pi/2, \pi/2]$ and speed $v \in [0, 10]$. Then, we manipulate these quantities as follows to obtain the feature vectors:

$$\begin{aligned}
\mathbf{s}_k &= \begin{bmatrix} 0 & 0 & v & 0 \end{bmatrix}^T, \\
\mathbf{s}_g &= \begin{bmatrix} r\cos\theta & r\sin\theta & 0 & \psi\mathrm{sign}(\theta) \end{bmatrix}^T,
\end{aligned}$$

where the sign$(\cdot)$ function is defined as follows:

$$\mathrm{sign}(x) = \begin{cases} 1, & x \geq 0, \\ -1, & x < 0. \end{cases}$$

Notice that the region described by this form of state sampling is a cone in front of the car. That is, we limit ourselves to simple maneuvers to avoid situations that would require reverse direction such as U- and Y- turns. We trained NMPC-Net on a dataset consisting of 500,000 samples.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 5)                11
 n)

 dense (Dense)               (None, 64)                384

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 16)                528

 dense_3 (Dense)             (None, 16)                272

 dense_4 (Dense)             (None, 8)                 136

 dense_5 (Dense)             (None, 2)                 18

=================================================================
Total params: 3,429
Trainable params: 3,418
Non-trainable params: 11
_________________________________________________________________
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 5)                11
 n)

 dense_16 (Dense)            (None, 16)                96

 dense_17 (Dense)            (None, 16)                272

 dense_18 (Dense)            (None, 16)                272

 dense_19 (Dense)            (None, 8)                 136

 dense_20 (Dense)            (None, 2)                 18

=================================================================
Total params: 805
Trainable params: 794
Non-trainable params: 11
_________________________________________________________________
```

Figure 2: Architecture for the largest tested neural network (left) and smallest tested neural network (right).

2.2 NMPC-Net Architecture

We recognized that the full feature dimensions (current and target states) aren't necessary to determine the optimal control. Instead, only the position (x, y) and heading ψ of the target state $\mathbf{s}_g$ relative to the current state $\mathbf{s}_k$ is needed. This reduces the input dimension of the problem from $\mathbb{R}^8$ to $\mathbb{R}^5$ (the current velocity and the relative final state). This feature dimension reduction resulted a smaller network, significantly decreasing the number of samples needed in the training dataset to span the same input space.

We designed out network as a dense feed-forward network with an input dimension of 5, and an output dimension of 2. Our goal is to develop a network that can efficiently and quickly approximate the NMPC solver, so we observed a tradeoff related to the neural network's depth and width: The deeper and wider the network, the more accurately it approximated the NMPC solver, the shallower and narrower the network, the faster it could be evaluated. We trained and tested several network sizes, The smallest and largest architectures are shown in Figure 2 and have 3,418 and 796 tuning parameters, respectively.

Note that we normalized the neural network input after the feature dimension reduction described previously with a normalization layer. Additionally, following common practice, we chose layer sizes in powers of 2 in order to maximize cache hit performance.

3 Results

In this section we describe how the developed controllers are used in the simulation of the kinematic bicycle model (1). Then, we review and discuss the performance of ζ_{NN} compared to that of ζ_{MPC}.

3.1 Simulation Method

We use the `RK45` integration method from the `scipy` Python3 package to to evaluate the tracking performance of each controller for the continuous-time kinematic bicycle model (1). This method allows us to simulate the system dynamics forward in time with more accuracy than that achieved by the forward-Euler approximation shown in (2). We can now approximate the state trajectory $\mathbf{x} : [t_0, t_f] \to \mathbb{R}^4$ which solves the following *initial value problem*:

$$\dot{\mathbf{x}}(t) = f\big(\mathbf{x}(t), \zeta(\mathbf{x}(t), \mathbf{s}_g)\big), \quad t \in [t_0, t_f], \tag{4a}$$

$$\mathbf{x}(t_0) = \mathbf{s}_0, \tag{4b}$$

where ζ is a *controller*, $\mathbf{s}_0 \in \mathbb{R}^4$ is the initial state of the system (i.e., at $t = 0$) and $\mathbf{s}_g \in \mathbb{R}^4$ is the constant goal state we wish to steer the system to.

Let $\mathbf{x}^{\text{MPC}}(t)$ be the solution to the initial value problem (4) when using the NMPC ζ_{MPC}, initial state $\mathbf{s}_0$ and goal state $\mathbf{s}_g$. Similarly, let $\mathbf{x}^{\text{NN}}(t)$ be the solution to the same initial value problem (4), but using the neural network controller ζ_{NN}. Then, we evaluate the performance of the NN controller ζ_{NN} with the Integrated Absolute Error (IAE) defined as:

$$\text{IAE} = \frac{1}{t_f - t_0} \int_{t_0}^{t_f} \|\mathbf{r}^{\text{NN}}(t) - \mathbf{r}^{\text{MPC}}(t)\|_2 \, \mathrm{d}t, \tag{5}$$

where $\mathbf{r} \triangleq (x, y)^T$ is the position vector of the vehicle (given by the first two elements of $\mathbf{x}^{\text{NN}}(t)$ and $\mathbf{x}^{\text{MPC}}(t)$.) Additionally, we also consider the simulation time for each controller as a performance metric. That is, we wish for ζ_{NN} to minimize both the IAE and simulation time.

3.2 Simulation Examples

We used a nearest-neighbor controller to validate our dataset and feature-reduction procedure. This controller, given a feature, finds the label associated with the feature that is closest (in the 2-norm $\|\cdot\|_2$ sense) from the current velocity and relative final state of the given feature. This controller was only used to validate that the generated data set and absolute to relative coordinate conversion was functioning. Thus, we show no simulation results or performance for this controller.

Figure 3 shows the results of the simulation using our large neural network controller along with the reference trajectory generated by the NMPC controller. The first simulation had an IAE of 0.160, while the second simulation had an IAE of 1.188. The neural network was able to predict the control input in 0.03s on average for both simulations, while the NMPC solver took 0.1s on average.

Figure 4 shows the results of the simulation using our small neural network controller, with the same initial and target states as the simulations for the large neural network. The first simulation had an IAE of 1.412, significantly larger than the IAE of the same simulation with the large neural network. The second simulation had an IAE of 0.615, smaller than the IAE of the large neural network for the same simulation. Surprisingly, the small network, despite having only 23.5% the number of parameters as the large network, took on average

0.03s per prediction as well. We discuss a hypothesis and why we think this could be improved in the discussion section.

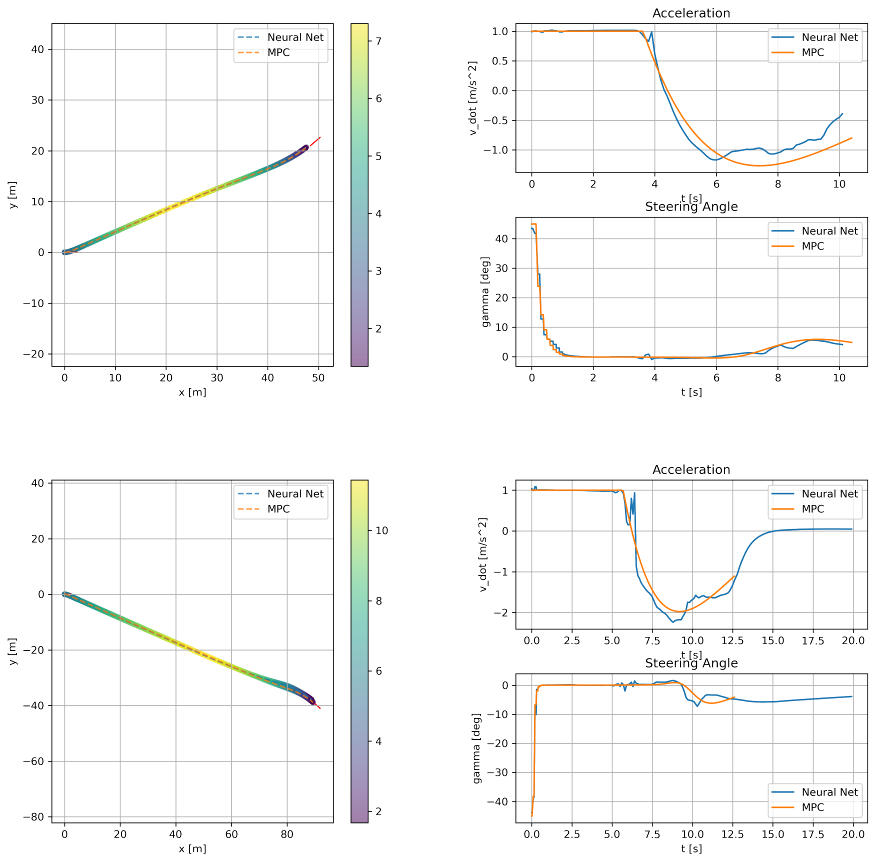


Figure 3: Large neural network controller simulations. The left plots show the path and speed profile of the kinematic bicycle model simulated with NMPC-Net (dashed-blue line), and NMPC (dashed-orange line). The right plots show the input profiles $\mathbf{u}(t)$.

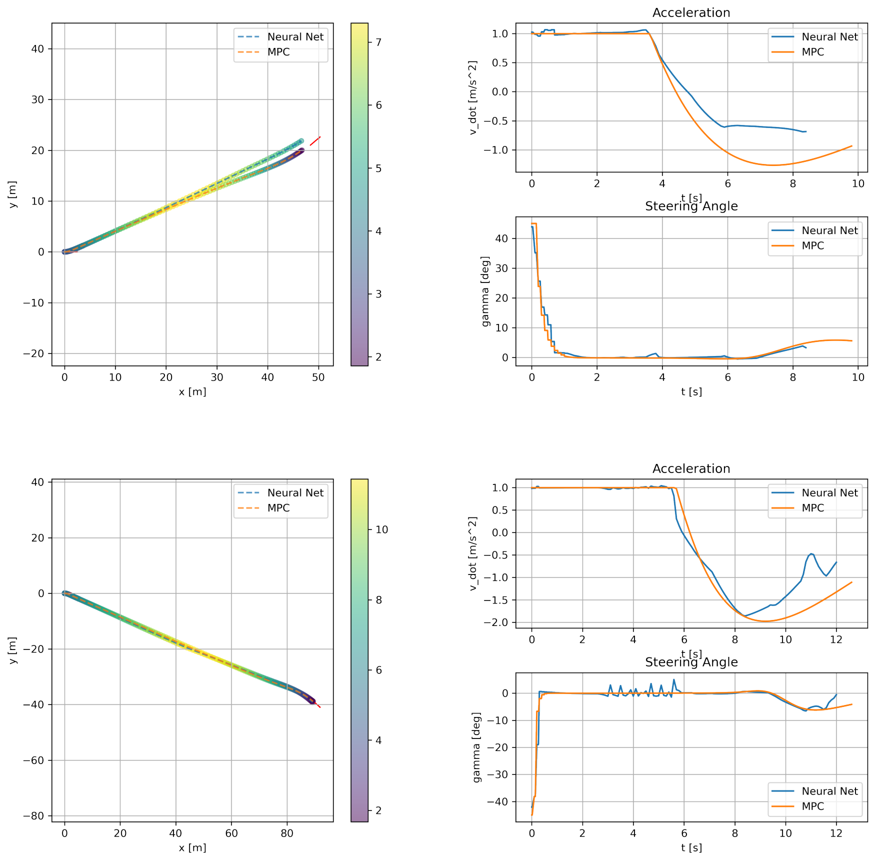


Figure 4: Small neural network controller simulations. The left plots show the path and speed profile, the right plots show the input profiles $\mathbf{u}(t)$. Both simulations run with the same initial and target states as in Figure 3.

4 Discussion

The simulation results shown in figures 3 and 4 clearly demonstrate the feasibility of our approach to using a neural network to replace a full nonlinear model predictive control solver. Not only are we able to successfully predict the control input that the NMPC control calculates to a high accuracy, but we do so in only 30% of the time.

As noted previously, the prediction time for the large and small networks was quite similar, both averaging to 0.03s per prediction, despite the small network having 23.5% the number of parameters as the large network. This suggests there is a large (relative to the actual time of doing the prediction) overhead in running a single prediction that we may be able to remove. In addition, the nonlinear programming solver used to solve the NMPC problem (IPOPT [2]) cannot be run on highly parallel architectures, while NMPC-Net could be deployed to edge TPU devices in an embedded system, enabling highly parallel computation. This could also result in greater performance though transfer overhead may be significant for such a small model.

Though we have demonstrated the feasibility of our approach, we restricted our target states to be within a cone in front of the vehicle. Nonetheless, NMPC problem can handle target states behind the vehicle. To further demonstrate the feasibility of our approach, we could expand the subset of the state space that we used for training.

References

[1] Victor Freire Ian Ruh. *ECE539_FinalProject*. `https://github.com/freiremelgiz/ECE539_FinalProject`. 2022.

[2] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[3] Michael L. Bynum et al. *Pyomo–optimization modeling in python*. Third. Vol. 67. Springer Science & Business Media, 2021.

[4] William E Hart, Jean-Paul Watson, and David L Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[5] Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[6] Guillaume Bellegarda and Quan Nguyen. "Dynamic Vehicle Drifting With Nonlinear MPC and a Fused Kinematic-Dynamic Bicycle Model". In: *IEEE Control Systems Letters* 6 (2021), pp. 1958–1963.

[7] Philip Polack et al. "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" In: *IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 812–818.

[8] Dongbing Gu and Huosheng Hu. "Neural predictive control for a car-like mobile robot". In: *Robotics and Autonomous Systems* 39.2 (2002), pp. 73–86.

[9] Abdul Afram et al. "Artificial neural network (ANN) based model predictive control (MPC) and optimization of HVAC systems: A state of the art review and case study of a residential HVAC system". In: *Energy and Buildings* 141 (2017), pp. 96–113.

[10] Julian Nubert et al. "Safe and fast tracking on a robot manipulator: Robust mpc and neural network control". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3050–3057.

[11] Maria Luiza Costa Vianna, Eric Goubault, and Sylvie Putot. "Neural Network Based Model Predictive Control for an Autonomous Vehicle". In: *arXiv preprint arXiv:2107.14573* (2021).

[12] Richard B Vinter and RB Vinter. *Optimal control*. Springer, 2010.