

Approximating MPC Solutions Using Deep Neural Networks: Towards Application in Mechatronic Systems

Edward Kikken, Jeroen Willems, Branimir Mrak and Bruno Depraetere
Flanders Make, Lommel, Belgium

Keywords: Optimal Control, Motion Control, Trajectory Planning, Artificial Neural Networks, Mechatronics.

Abstract: Model Predictive Control is an advanced control technique that can yield high performance, but it is often challenging to implement. Especially for systems with dynamics that are complex to model, have strong non-linearities, and/or have small time constants, it is often not possible to complete the needed online optimizations fast and reliable enough. In this work we look at approximating the MPC solutions using black-box models i.e. deep neural networks, so that the computational load at runtime is strongly reduced. We use a supervised learning approach to train these models to yield outputs similar to those of an example dataset of offline pre-computed MPC solutions. We illustrate this approach on three realistic (active-suspension system, parallel robot, and a truck-trailer), illustrating the typical workflow and how the approach has to be set up to address the varying challenges. We show that the approximate MPC solutions yield a high level of performance, reaching nearly the level of the original MPC, yet at a strongly reduced computational load.

1 INTRODUCTION

There is an increasing need for advanced control, due to stricter requirements for accuracy, productivity and/or energy efficiency, as well as systems becoming more complex and being used in more variable conditions. One option to achieve this is Model Predictive Control (MPC). This is a powerful technique that, during operation, at every time step, chooses the best control action by solving a numerical optimization problem, using a model for the system dynamics to evaluate the impact of the different control choices on the constraints and the cost function (Rawlings, 2000). One of the key drawbacks of MPC is the computational burden associated with the aforementioned optimization. As a result, MPC becomes too slow and/or too difficult to run on industrial controllers, especially for (i) systems with small time constants, (ii) systems with many states, and/or (iii) systems with highly non-linear dynamics (for which MPC is often denoted NMPC).

This can be resolved in two different manners. Firstly, the MPC can be improved, by e.g., simplifying the models, or by improving the used optimization routines so they converge more quickly and more reliably (Vanroye et al., 2023). Secondly, and what this paper will focus on, is to not solve the MPC itself during machine operation, but to instead pre-calculate and

approximate its solution, so that during machine operation the calculation of the control does not involve the solving of an online optimization problem.

Already several works have looked at such approaches for pre-calculating MPC solutions. Historically, explicit MPC was developed in (Bemporad et al., 2000). In this and later works, it was shown that for an MPC applied to a system with affine dynamics and constraints, as well as a quadratic cost, the control output is a piecewise affine function of the states. The MPC solution is then essentially equal to a set of linear controllers, depending on the current system state. And crucially, these can all be pre-calculated and stored in a big look up table. The main drawbacks of this approach are (i) the exponential scaling of the number of needed regions as a function of the problem size, and (ii) the lack of support for general non-linear dynamics. Due to the first, approximations of non-linear dynamics cause explicit MPC to quickly become non-tractable.

In an attempt to address these issues, several works have looked at approximating the MPC solutions, instead of trying to exactly describe them. Approximations have been built using for example polynomials (Kvasnica et al., 2011) or multi-scale basis functions (Summers et al., 2011). Recently, neural networks have also been considered. For example, combined with a learning scheme to adapt weights and

biases to capture optimal control outputs (Parisini and Zoppoli, 1995), and MPC solutions (Winqvist et al., 2021; Chen et al., 2018). Some more recent works have switched to using supervised learning, to directly learn from examples generated using an MPC or NMPC controller offline (Hertneck et al., 2018; Nubert et al., 2020; Lucia et al., 2021). In recent works such an approach has been applied to non-linear systems as well as linear ones (Hertneck et al., 2018; Lucia and Karg, 2018). Several works have also looked at describing or bounding the accuracy or suboptimality of the solutions (Hertneck et al., 2018), trained secondary networks (Zhang et al., 2019) to indicate when suboptimality or even infeasibility becomes an issue, or proposed architectures relying on cloud-based supervision (Adamek and Lucia, 2023). Recent work has also looked at fine-tuning such methods after deployment (Hose et al., 2024).

Such Approximate MPC or A-MPC has been applied to various applications, including building HVAC (Karg and Lucia, 2018), a polymerization (Lucia and Karg, 2018) and a stirred tank reactor (Hertneck et al., 2018), step down (Maddalena et al., 2020) and resonant power convertors (Lucia et al., 2021), robots (Nubert et al., 2020), and battery charging (Pozzi et al., 2022). Our contribution is to apply it to several more and diverse examples: one straightforward, one with a finite task and an economical cost, and one on path planning with two families of differing solutions. These later two introduce challenges not tackled in the existing examples, and we propose several A-MPC changes to handle those. To tackle the finite tasks we provide the A-MPC with extra information on the remaining task, allowing the network to adapt during the execution. To tackle the path planning we treat the A-MPC as a mixed integer control problem, and add a classifier for the integer values. This differs from existing approaches like using deeper networks (Karg and Lucia, 2018), or using stochastic (Bernoulli) layers to handle integer variables (Okamoto et al., 2024). Furthermore we will look into the trade-off between the number of MPC training examples and the resulting A-MPC performance, since in practice generating data for training these A-MPC takes significant time. Detailed statistical analysis of robustness for these applications will however be left to future work.

2 PRELIMINARIES

In this section, we first introduce the mathematical notation considered in this paper, and then give a generic MPC formulation, on which we will build further.

2.1 Notation

We denote $\mathbb{R}^n$ as the set of real vectors of dimension n , and $\mathbb{R}^{m \times n}$ as the set of real matrices of dimension $m \times n$. A control sequence $[u_1 \ u_2 \ \dots \ u_N]^T \in \mathbb{R}$ is denoted by the bold vector $\mathbf{u}$ with $N \in \mathbb{N}$ discrete samples. A specific sample is given by integer $\cdot(k)$ with $k \in \mathbb{N}$.

2.2 Generic Model-Predictive Control (MPC)

Model Predictive Control (MPC) is a generic class of control methods which utilize a dynamic model to predict the future response of the system, and choose the control actions yielding the best predicted response (given a cost function and constraints). To do so, at each control interval in a given finite-time horizon (of N samples), MPC optimizes current and future controls and state trajectories (Qin and Badgwell, 2003). Once the optimal controls are found, the first values are applied at the current time-step, the system's response is observed, and the MPC is optimized again starting from the observed system state. This procedure is repeated at every time step, which allows the MPC to react in a manner similar to feedback control, to deviations due to e.g., model-plant mismatch and disturbances. Using the predictive approach it also becomes straightforward to exploit preview or predictions of upcoming events like disturbances or external loads.

As said, the MPC relies on a model. Here, we consider a discrete-time model:

$$\mathbf{x}(k+1) = f(\mathbf{x}(k), \mathbf{u}(k)), \quad (1)$$

with the dynamic states $\mathbf{x}(k) \in \mathbb{R}^{n_x}$ and the control actions $\mathbf{u}(k) \in \mathbb{R}^{n_u}$, with n_x and n_u the number of states and inputs respectively. In the above, f denotes the state propagation function (e.g., ordinary differential equation (ODE)). This generic function can be linear or non-linear, SISO or MIMO, etc.

This model can then be used to set up the MPC formulation. Here, we use a horizon of $N \in \mathbb{N}$ intervals (samples), yielding:

$$\underset{\mathbf{x}(\cdot), \mathbf{u}(\cdot)}{\text{minimize}} \quad \mathcal{J}(\mathbf{x}(\cdot), \mathbf{u}(\cdot)), \quad (2a)$$

$$\text{s. t.} \quad (1) \quad \forall k \in [1, N], \quad (2b)$$

$$\underline{\mathbf{x}} \leq \mathbf{x}(k) \leq \bar{\mathbf{x}} \quad \forall k \in [1, N+1], \quad (2c)$$

$$\underline{\mathbf{u}} \leq \mathbf{u}(k) \leq \bar{\mathbf{u}} \quad \forall k \in [1, N], \quad (2d)$$

$$\underline{\mathbf{g}} \leq \mathbf{g}(\mathbf{x}(\cdot), \mathbf{u}(\cdot), \mathbf{y}(\cdot), \mathbf{p}) \leq \bar{\mathbf{g}}. \quad (2e)$$

with cost function $\mathcal{J}(\mathbf{x}(\cdot), \mathbf{u}(\cdot))$. Matrix $\mathbf{x}(\cdot) \in \mathbb{R}^{n_x \times N+1}$ contains the optimization variables over the entire horizon for the differential states: $\mathbf{x}(\cdot) = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N, \mathbf{x}_{N+1}]$, and $\mathbf{u} \in \mathbb{R}^{n_u \times N}$ similarly contains all control actions. In the above, (2a) implements the cost function, (2c) and (2d) implement lower and upper bounds on the state and input variables, and (2e) implements miscellaneous constraints (e.g., initial condition and path constraints).

As mentioned previously, MPC solves the optimization problem denoted in (2) at every time step. After solving, the first sample of the computed input sequence $\mathbf{u}(\cdot)$ is then applied to the system, i.e., $\mathbf{u}(1)$, and we again solve the optimization problem for the next time sample.

3 PROPOSED APPROACH

Our proposed approach to approximate MPC solutions using black-box models, namely deep neural networks (DNN) consists out of three steps:

1. Offline, we formulate the MPC problem (see Section 2.2) for the use-case at hand and solve it for a large combination of states, constraints, possibly model-variants, etc. The goal is to capture the desired operating region of the system / task. The result is an input dataset $\mathcal{X}$ containing the system states and additional features like e.g. obstacle locations or the time left to task completion, and an output dataset $\mathcal{U}$ containing the optimal control actions as computed by the MPC.
2. Still offline, DNN is trained that aims to approximate the mapping from input dataset $\mathcal{X}$ to output dataset $\mathcal{U}$.
3. At run-time, DNN is used to calculate the plant input as a function of the current input features, and aims to act as a direct replacement to the original MPC controller.

3.1 MPC Problem Formulation and Dataset Generation

In order to set-up the MPC for a given application, we use the template from Section 2.2 and implement application-specific costs and constraints. We use CasADi (Andersson et al., 2019) to set-up the models and optimization problems, and the solver IPOPT (Biegler and Zavala, 2009) is used to efficiently solve the problem.

As mentioned previously, the goal is to create two datasets:

- Input dataset $\mathcal{X} \in \mathbb{R}^{N_x \times M}$ containing $N_x \in \mathbb{N}$ features, such as states, as well as additional features like current sample in the task and time to completion, potentially a preview of upcoming disturbances or other external signals relevant to the controller, locations of obstacles, etc. $M \in \mathbb{N}$ denotes the total number of MPC solutions in the dataset.
- Output dataset $\mathcal{U} \in \mathbb{R}^{N_u \times M}$ containing the optimal control actions (e.g. torques, forces, trajectories). $N_u \in \mathbb{N}$ denotes the number of outputs.

The goal is to generate a rich dataset, covering the states, inputs, etc. of interest. The richer the datasets, the better the approximate controller will perform in a variety of unseen conditions, but the higher the computational effort required to build the dataset and to train the DNN. To address this, we followed an iterative approach for building our dataset. We first defined a bounded region of realistic values for states, inputs, etc. Then, we started with a relatively compact dataset of MPC solutions i.e., $\mathcal{X}$ and $\mathcal{U}$, sampled from the complete region of operation and used these to fit a DNN. Afterwards, we analyzed the performance on a wide set of test points $\mathcal{X}_{val}$ within the desired region of operation but which are not included in $\mathcal{X}$. If the performance on $\mathcal{X}_{val}$ is not yet sufficient more points were added thereby increasing the density of $\mathcal{X}$, until the performance is sufficient. This was done in ad-hoc manner since this paper focuses on the applications, but more thorough statistical analysis of performance for training set size has been performed in e.g. (Hertneck et al., 2018). We also used random samples of states or trajectories through the operation space. Through optimal design of experiments (DoE), a more targeted choice could be made regarding this trade-off, like in (Gupta et al., 2023; Chen and Peng, 2017).

3.2 Training Deep Neural Networks

A DNN is trained to fit the mapping from input dataset $\mathcal{X}$ to output dataset $\mathcal{U}$. Various design choices can be made, e.g., the type of network (DNN, possibly in tandem with additional networks as classifiers), the corresponding architecture and its hyperparameters. Typically, the appropriate choice of the architecture follows from the complexity of the considered use-case.

In the most basic format, we consider a feedforward neural network, because of its general purpose nature. The neural network architecture includes an input layer with N_x input features, N_{layers} fully-connected hidden layers each containing N_{hidden} neurons, and an output layer with N_u output features representing the control actions. All layers are fully connected. In

between the layers, we use activation functions, e.g., ReLU or Tanh. A simple example of such a neural network is shown in Figure 1.

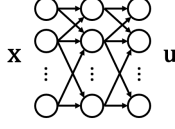


Figure 1: A simple multi-layer feedforward neural network mapping inputs to outputs.

We implement DNN models using Torch (Paszke et al., 2019) and use the Adam optimizer (Kingma and Ba, 2014) to optimize the weights and biases. The cost function used to train the DNN is the mean squared error between the output features in $\mathcal{U}$ and the model's predictions. After scaling the input and output datasets, the training is ran for a number of epochs until the fitting error is sufficiently low.

3.3 Deployment

After the DNN has been trained using the approach in Sections 3.1 and 3.2, it can be deployed on the machine controller. There it receives online measurements and computes its approximate MPC control actions, thereby replacing the original MPC.

4 VALIDATION

In this section, we validate the proposed algorithm on a range of use-cases:

- An active-suspension system, where the suspension is controlled to optimize comfort, while subject to road variations. This is a simple linear case to start with, with a relatively small timescale, and a classical quadratic cost of states and control actions.
- A parallel SCARA robot, executing energy-optimal point-to-point motions while avoiding an obstacle. This is a more complex case, with non-linear dynamics, a lower time constant, and a finite task to be completed.
- A truck-trailer, which has to make a U-turn in a tight space. This is an even more challenging case, due to the non-linear dynamics, and since the (integer) numbers of reversals is not the same for all possible turns.

The performance difference between the two approaches can be addressed on multiple fronts:

- Performance in terms of cost and constraints. The results can be compared to the original MPC ones,

initially on training data to study the impact of fitting errors, and then on validation data.

- Computational efficiency. The goal of the approach is to reduce the computational effort required during run-time. Nevertheless, the proposed method requires an increased pre-calculation cost for building the training set and training the network. Here we use the needed calculation time for both, but ideally this should be done based on FLOPS.

For each of the examples, we assume all states are measurable without noise and / or disturbances. In practice, it can be needed to add estimators to obtain these, like for most MPC implementations.

4.1 Active-Suspension System

In this section, we consider the active suspension use-case. The suspension system can be controlled to manage the vertical movement of the car. Unlike passive suspension systems, which rely on fixed springs and dampers, active suspension systems are able to adjust in real-time to the road conditions and driving dynamics, thereby improving passenger comfort. The considered system is modeled as a quarter car, shown schematically in Figure 2. It consists of a car body and a suspension, driving on a given road.

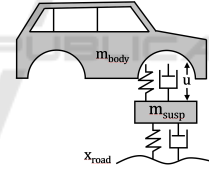


Figure 2: The considered quarter car with active suspension.

The system has 5 states:

$$\mathbf{x} = [x_{body} \quad v_{body} \quad x_{susp} \quad v_{susp} \quad x_{road}] \in \mathbb{R}^5,$$

and 2 inputs:

$$\mathbf{u} = [u_{susp} \quad v_{road}] \in \mathbb{R}^2,$$

where x , v and u denote displacement, speed and input force respectively. The linear dynamics are given in ODE format as:

$$\begin{aligned} \dot{x}_{road} &= v_{road}, \\ \dot{x}_{body} &= v_{body}, \\ \dot{x}_{susp} &= v_{susp}, \\ \dot{v}_{body} &= \frac{1}{m_{body}}(-F_{susp} + u_{susp}), \\ \dot{v}_{susp} &= \frac{1}{m_{susp}}(+F_{susp} - u_{susp} + F_{road}), \end{aligned} \quad (3)$$

where:

$$F_{susp} = d_{susp}(v_{body} - v_{susp}) + k_{susp}(x_{body} - x_{susp}),$$

and:

$$F_{road} = d_{tire}(v_{road} - v_{susp}) + k_{tire}(x_{road} - x_{susp}).$$

The model parameters are shown in Table 1.

Table 1: Model parameters for the active suspension.

Parameter	Value	Unit	Description
m_{body}	625	kg	Body mass (25%)
m_{susp}	320	kg	Mass of suspension
d_{susp}	0	Ns/m	Damping constant of suspension
d_{tire}	3755	Ns/m	Damping constant of wheel and tire
k_{susp}	80000	N/m	Spring constant of suspension
k_{tire}	125000	N/m	Spring constant of wheel and tire

The control goal of the MPC is to minimize the displacement of the car body, x_{body} , by controlling u_{susp} . Therefore, the cost function (Eq. 2a) is set to:

$$\mathcal{J} = w_x \sum_{i=1}^{i=N} x_{body}(i)^2 + w_u \sum_{i=1}^{i=N} u_{susp}(i)^2.$$

Scalar weights $w_x \in \mathbb{R}$ and $w_u \in \mathbb{R}$ trade off the two cost functions: minimization of displacement and input regularization respectively. The constraints are given by:

- Initial condition: $\mathbf{x}_1 = \mathbf{x}_{current}$, i.e., the initial state of the optimization problem is set to the current state of the system.
- Input constraint: $-3000 \leq u_{susp} \leq 3000$.

We consider a sampling time of 0.04 s and an MPC horizon of 0.2 seconds ($N = 5$ samples). A forward Euler integrator is used to integrate the dynamics. The MPC receives preview of the road profile (x_{road} and v_{road}) 5 samples ahead.

After setting up the MPC, it is run for a grid of conditions to construct input dataset $\mathcal{X}$ to output dataset $\mathcal{U}$, which contain the following features:

- The input features are selected as the 5 states at the current sample, and 5 samples ahead (preview) of v_{road} .
- A single output feature is considered: the next sample of u_{susp} .

In total, 1000 MPC solutions are generated by varying the input features as follows:

- The first 4 states are excited at $[-0.3 \ 0 \ 0.3]$, and the last state at $[-0.1 \ 0 \ 0.1]$.
- The road profile, v_{road} , is excited at $[-3 \ 0 \ 3]$ for each of the samples.

Note that a full factorial sampling grid is considered, but that the coverage is relatively sparse. Due to the linear dynamics of the system, the model trained next performs well at intermediate points, removing the need for additional intermediate samples in the training data.

The considered neural network is designed as a feed-forward neural network consisting of 2 fully connected hidden layers with 20 neurons each, with ReLU activation functions. After training, a root-mean-squared error of 18 N is achieved on the training data (0.85%). Next, the result is tested / deployed for a given road profile with a length of 50 samples (10 times the MPC horizon), and benchmarked with the MPC solutions. In Figure 3, the results are shown. It can be seen that the solution of the proposed approach closely matches that of the original MPC, also if the input constraints are active. In terms of the cost, the original MPC has a cost of 0.127, while the MPC-approximation has a 0.5 percent higher cost. Regarding evaluation time: evaluating the original MPC takes 33 ms per iteration, while the proposed A-MPC approximation only requires 0.5 ms per iteration, which is limited by our non real-time operating system and is expected to be significantly faster on a typical embedded controller.

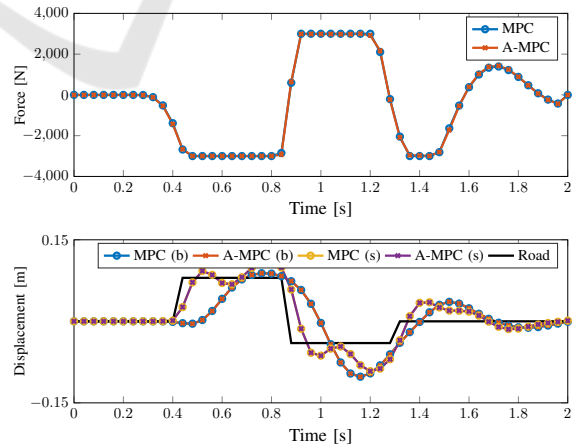


Figure 3: Validation on a given road profile. (b) denotes the body, (s) denotes the suspension.

4.2 Parallel SCARA

In this section, we consider a parallel SCARA robot, shown in Figure 4. Such a system is often used for pick-and-place or assembly operations, where high speed and accuracy are required. The considered system is driven by two motors, and controls the location of an end-effector. The system has non-linear dynamics and is MIMO, making it challenging to control.

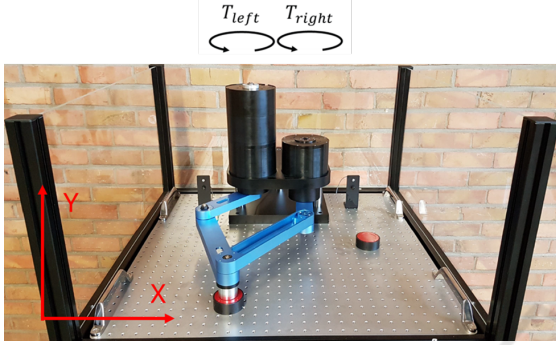


Figure 4: The considered SCARA setup.

The system is driven by two motors (left and right), yielding motor torques $\mathbf{u} = [T_{left} \ T_{right}] \in \mathbb{R}^2$, and controls the position of an end-effector, yielding states $\mathbf{x} = [x_{ee} \ \dot{x}_{ee} \ y_{ee} \ \dot{y}_{ee}] \in \mathbb{R}^4$. In this paper, we consider a simulation study, based on a model designed using ROBOTRAN (Docquier et al., 2013). For the sake of brevity, we refer to (Singh et al., 2024) for the detailed model equations.

For this case, the control goal of the MPC is to conduct a point-to-point motion, from a given initial end-effector state ($\mathbf{x}_{init} = [x_{init} \ 0 \ y_{init} \ 0]$) to a final end effector state ($\mathbf{x}_{end} = [x_{end} \ 0 \ y_{end} \ 0]$), while minimizing the required input torque. Therefore, the cost function (Eq. 2a) is set to minimize the torques:

$$J = \sum_{i=1}^{i=N} T_{left}(i)^2 + \sum_{i=1}^{i=N} T_{right}(i)^2. \quad (4)$$

The constraints are given by:

- Initial condition: $\mathbf{x}_1 = \mathbf{x}_{current}$, i.e., the initial state of the optimization problem is set to the current state of the system. For the first sample of the task, $\mathbf{x}_1 = \mathbf{x}_{init}$.
- Obstacle avoidance: the end-effector has to stay outside of a given constraint surface: a circle with a radius of 0.05 m, illustrated later.

There is also the constraint to ensure the robot reaches the desired target $\mathbf{x}_{end}$ at the end of the task length, which is 0.25 s. To do so, we re-solve the MPC every sample with a sampling time of 0.0025 s, and as the task is executed, each time, the length of the MPC

horizon N is decreased by 1, going from 100 initially down to 1 as the task is completed.

We consider the following features for input dataset $\mathcal{X}$ to output dataset $\mathcal{U}$:

- The input features are selected as the current states $\mathbf{x}_{current}$, the desired final positions of the end-effector, x_{end} and y_{end} , and finally also the (integer) number of samples until the end of the total task horizon. This last feature provides a notion of how many samples are left until the end of the task.
- Two output features are considered: the motors torques T_{left} and T_{right} .

The datasets $\mathcal{X}$ and $\mathcal{U}$ are constructed by varying the initial and final condition (end-effector positions) of the point-to-point motion. We vary the initial and final x-position between -0.23 and 0.23 m in 8 steps. Similarly, we vary the y-position between 0.20 and 0.34 m in 8 steps, yielding a number of 4096 configurations. Each of the configurations has a task length of 100 samples, so in total we have 409600 MPC solutions. All trajectories, including the obstacle, are visualized in Figure 5 where it can be seen that the training data covers the region of interest densely.

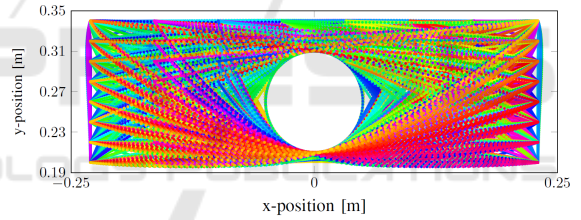


Figure 5: The generated trajectories, including the obstacle.

Again, we consider a feed-forward neural network, but in this case it consists of 2 fully connected hidden layers with 150 neurons, with ReLU activation functions. After training, a root-mean-squared error of 0.14 Nm is achieved on the training dataset.

In order to test the proposed approach we consider two cases. First, we will consider an initial and final point within the training data. Second, we consider an intermediate point, to investigate how well the algorithm extrapolates to unseen conditions.

The first case considers $x_{init} = 0.23$, $y_{init} = 0.28$ and $x_{end} = -0.23$, $y_{end} = 0.2$, which are direct members of the training data. With respect to the cost function, the original MPC has an RMS torque of 0.82 Nm and the A-MPC an RMS torque of 0.83 Nm.

The second case considers $x_{init} = -0.13$, $y_{init} = 0.33$ and $x_{end} = 0.13$, $y_{end} = 0.23$. In this case, the initial and final conditions are placed relatively far away from the grid points in the training dataset, aiming to investigate how well the proposed algorithm general-

izes to unseen data. In Figure 6, the results are shown. In the figure, it can be seen that the solutions are still very similar. With respect to the cost function, the original MPC has an RMS torque of 0.660 Nm and the A-MPC an RMS torque of 0.662 Nm.

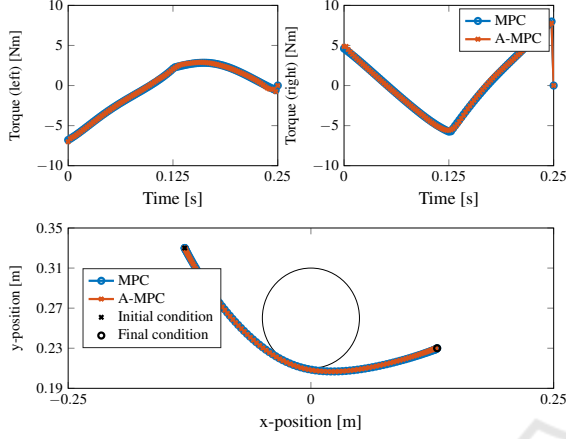


Figure 6: Result for conditions *outside* the training set.

Next, we study the effect of measurement noise on the performance. To do so, a white noise disturbance is added to the states (magnitude of 0.5 mm (position) and 1 mm/s (velocity) respectively). For each of the controllers, the same noise profile is applied. The results are shown in Figure 7. The following observations are made:

- When noise is added, both the MPC as well as the A-MPC deviate from the original solution.
- The obtained input / displacement profiles are relatively similar, but the input signal of the MPC has more oscillations (especially when it nears the circular constraint). The A-MPC does not display this effect, since these effects were not accounted for in the training data.
- The computed costs for both solutions are the same, 0.665 Nm. However, both solutions do slightly violate the constraints. This will be further dealt with in future work: by designing an MPC robust to noise, and by training an A-MPC on the resulting solutions.

Regarding evaluation time: evaluating the original MPC takes 40 ms per iteration (on average), while the MPC approximation only requires 0.5 ms per iteration (as before limited by our non real-time system). Hence, the original MPC is on average 16 times slower compared to the sampling time of the system (and thereby does not meet the real-time requirements), compared to the proposed approach which is 5 times faster. Note however that it took approximately 4 hours to generate the dataset and train the model used for the A-MPC.

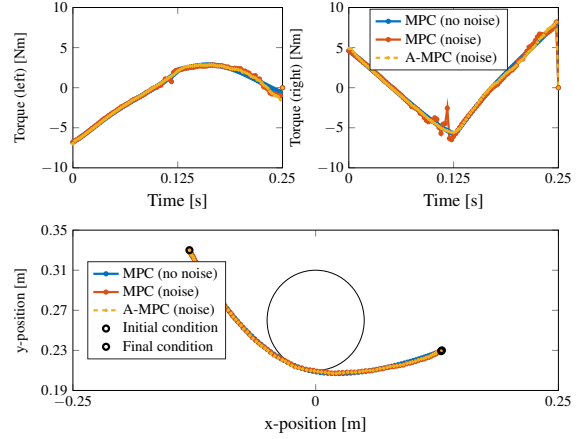


Figure 7: Result for conditions *outside* the training set, including noise.

To arrive at the presented A-MPC performance, we applied the iterative training data generation approach from Section 3.1. We varied the initial and final x and y positions in either 2, 4, 6 or 8 steps, yielding increasingly rich training data sets of 16, 256, 1024 and 4096 trajectories, respectively. After each data set generation we analyzed the root-mean-squared error of the torques predicted by the A-MPC obtained on those versus those of the original MPC, on a different validation dataset, and increased the number of steps further if accuracy was not yet sufficient. The result is shown in Figure 8. In the figure, we can see that denser input space sampling (and more trajectories in the training dataset), yields better performance on the validation set, as is to be expected.

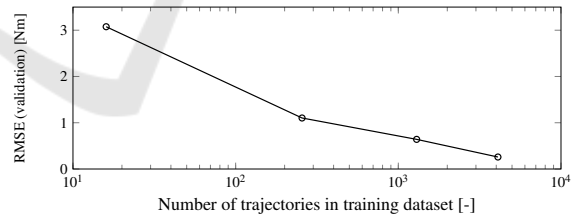


Figure 8: Trade-off between the number of trajectories in the dataset and the fitting performance on a validation dataset.

4.3 Truck-Trailer

The last case considered in this paper is a truck-trailer, which has to make a U-turn. Depending on the available space, it can sometimes be needed to make a number of speed reversals, but not always. An example without reversal is shown in Figure 9, and one with one speed reversal in Figure 10. For the latter, the truck moves from a given initial condition with the truck oriented to the top of the page, to a final condition with it oriented down, while subject to space

constraints.

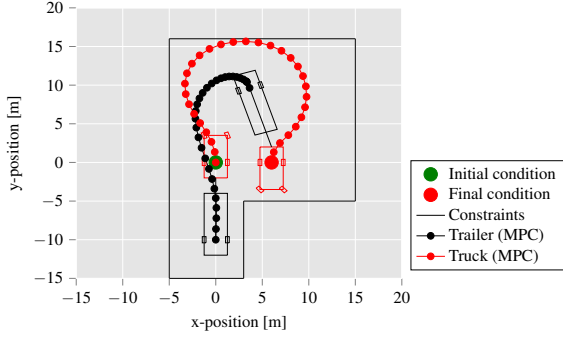


Figure 9: Example of paths for truck and trailer when performing a U-turn without a speed reversal.

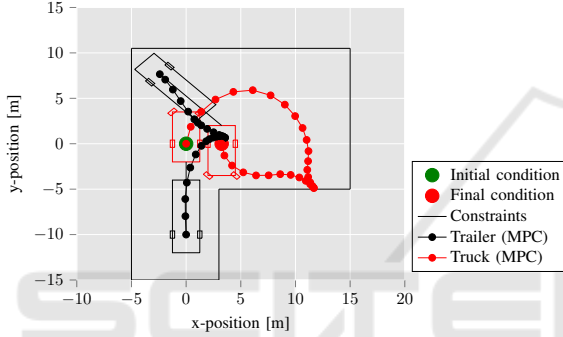


Figure 10: Example truck-trailer paths with one speed reversal.

In contrast to the previous use-cases discussed in this paper, we do not consider an MPC which is solved at each time step in the horizon. Instead, we construct a model that is able to predict the ideal trajectory (covering the entire horizon) at once. We do this since in reality we would then let an external path-tracking controller track the path, not an MPC, which is left out of scope for this paper.

The system has 4 states:

$$\mathbf{x} = [\theta_b \ x_a \ y_a \ \theta_a] \in \mathbb{R}^4,$$

and 2 inputs:

$$\mathbf{u} = [\delta_a \ v_a] \in \mathbb{R}^2,$$

where x and y denote displacements in their respective direction, v the velocity, θ denotes the heading and δ the steering angle input. Subscript $(\cdot)_a$ denotes the truck, and subscript $(\cdot)_b$ the trailer.

The non-linear dynamics are given in ODE format

as:

$$\begin{aligned} \dot{\theta}_b &= \frac{v_a}{L_b} \sin(\beta_{ab}) - \frac{M_a}{L_b} \cos(\beta_{ab}) \frac{v_a}{L_a} \tan(\delta_a), \\ \dot{x}_a &= v_a \cos(\theta_a), \\ \dot{y}_a &= v_a \sin(\theta_a), \\ \dot{\theta}_a &= \frac{v_a}{L_a} \tan(\delta_a), \end{aligned} \quad (5)$$

where $\beta_{ab} = \theta_a - \theta_b$. The model parameters are shown in Table 2.

Table 2: Model parameters for the truck-trailer.

Parameter	Value	Unit	Description
L_a	3.5	m	Truck: axle to axle distance
M_a	2	m	Truck: distance of hitch behind axle
W_a	2.5	m	Truck: width
L_b	8	m	Trailer: axle to hitch distance
M_b	2	m	Trailer: distance of axle to rear wall
W_b	2.5	m	Trailer: width

For this case, the control goal is to conduct a point-to-point motion, from a given initial state ($\mathbf{x}_{init} = [\frac{\pi}{2} \ 0 \ 0 \ \frac{\pi}{2}]$) to a final state ($\mathbf{x}_{end} = [(\cdot) \ x_{end} \ 0 \ -\frac{\pi}{2}]$). Note that the final trailer angle θ_b is left free.

The considered cost function is to minimize:

$$J = \sum_{i=1}^{i=N} v_a^2,$$

so the motion is completed in as smooth a manner as possible. We consider the following constraints:

- Initial and final condition, as mentioned previously.
- Steering angle constraint: $-\frac{\pi}{4} \leq \delta_a \leq \frac{\pi}{4}$.
- Relative angle: $-\frac{\pi}{2} \leq \beta_{ab} \leq \frac{\pi}{2}$.
- Truck speed: $-3 \leq v_a \leq 3$.
- Space constraints: $y_a < y_{max}$, $y_b < y_{max}$, $x_a > -5$, $x_b < -5$.

To solve the optimal control problem, we consider a sampling time of 1 s and a total task length of $N = 30$ samples. We solve the entire task at once.

In order to generate the input and output dataset, we vary $x_{end} \in \mathbb{R}$ between 3.25 and 6 m in 12 steps and $y_{max} \in \mathbb{R}$ between 8 and 16 m in 33 steps, yielding 396

solutions in total. The resulting truck trajectories ($\mathbf{x}_a$ and $\mathbf{y}_a$) are visualized in Figure 11. All solutions start from an initial condition of 0 in x - and y -direction. It can be seen that some solutions have 0 speed reversals, whereas other solutions have 1. Sometimes, the speed reversal occurs towards the beginning of the trajectory (causing the truck to move in negative y -direction) and sometimes towards the end. The amount of turns is affected by the settings y_{max} and x_{end} , further illustrated later on in this section.

Using this data, we then define the following features that will be used for training the A-MPC:

- Input features: x_{end} , y_{max} . Note that some of the initial and final states are fixed (eg also x_{end} varies) and thus omitted from the dataset, but in general these would also need to be included as input.
- Output features: the truck trajectories (with length N) $\mathbf{x}_a \in \mathbb{R}^N$, $\mathbf{y}_a \in \mathbb{R}^N$, $\mathbf{v}_a \in \mathbb{R}^N$.

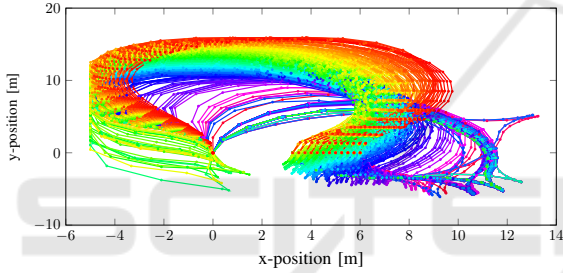


Figure 11: The trajectories of the truck, used for training.

An extra feature that is relevant here is the (integer) number of speed reversals (0 or 1) $n_{reversals} \in \mathbb{N}$. This makes the path planning problem a mixed integer control problem. Rather than solving this using deeper networks (Karg and Lucia, 2018), or using stochastic (Bernoulli) layers (Okamoto et al., 2024), we augment the neural network with a separate classifier.

To do so, we have chosen a binary classifier to first predict the number of speed reversals, based on input features: x_{end} and y_{max} . Hereby, for simplicity, this is done using a support-vector machine (SVM), employing an RBF kernel. In another application use of neural network classifiers could be preferred. The predicted $n_{reversals}$ by this classifier is used as an input to the next part of our A-MPC, which is a feed-forward neural network, consisting of 2 fully connected hidden layers with 1500 neurons, with ReLU activation functions. The overall interconnection is shown in Figure 12.

The result of this SVM is shown in Figure 13. For the training set, it is able to perfectly predict the number of speed reversals.

Regarding the neural network, it converges to a root-mean-squared error of 0.017 m for the displace-

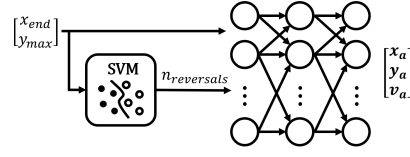


Figure 12: The interconnection of the neural network and SVM.

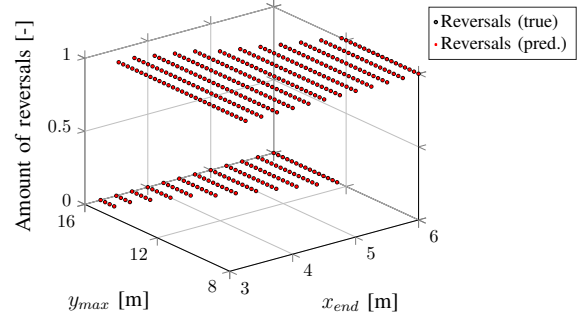


Figure 13: The amount of speed reversals (true and predicted), based on the input features.

ment in x - and y -direction (0.25%) and 0.003 m/s for the truck velocity (0.23%) on the training data.

Next, the approach is first validated on a trajectory for one of the grid points in the training data: $y_{max} = 16$ m, $x_{end} = 6$ m. For this case, no speed reversals are present, not in the optimal solution used for training and also not in the approximation. With respect to the trajectories in x - and y -direction, a root-mean-squared error of 0.01 m is achieved. Next, we consider a trajectory within the same ranges for y_{max} and x_{end} , but not on any of the grid points used for training, with $y_{max} = 9.625$ m, $x_{end} = 3.875$ m. Now there is one speed reversal for the approximation, which matches the outcome of the optimal solution that we find when we calculate it for validation (but this was not used for training), as shown in Figure 14. In this case, the root-mean-squared error is small as well: 0.02 m.

Regarding computational effort: solving the MPC takes on average 3 seconds, whereas inference of the DNN requires only 3 milliseconds.

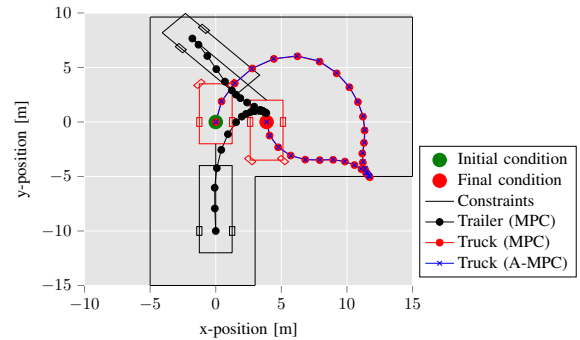


Figure 14: Validation on conditions *outside* the training set.

5 CONCLUSION

We have shown application of an approximate MPC technique on several challenging mechatronics cases. It relies on supervised learning, to train DNNs to match MPC examples. This allows to pre-calculate the MPCs, and thereby reduces the computational load at runtime. This makes this approach very usable for cases with complex or non-linear dynamics and/or small time constants, for which classical MPC would otherwise typically not be realistic. We have illustrated the versatility of the approach by applying it to several different examples. We have also shown the workflow for how to tailor the approach for each of those examples, including extensions for handling finite tasks and mixed integer control problems.

We have worked in a pragmatic manner, but in the future will work on (i) a thorough stochastic analysis of training set and optimality or feasibility, allowing to give stronger validations or even verifications of the A-MPC, (ii) more targeted procedures to generate training data, and (iii) different architectures, wherein approximations are used alongside classical methods, for example like in (Chen et al., 2022) where an MPC is given a feasible initialization using an efficient approximation.

While we have only reported needed training time and inference time, it is interesting for future work to study the ecological impact, trading off increased pre-processing cost with the reduced run-time cost like done in (Lacoste et al., 2019).

ACKNOWLEDGMENT

This research was supported by: Flanders Make, the strategic research centre for the manufacturing industry in Belgium, specifically by its DIRAC SBO and LearnOPTRA SBO research projects, and the Flemish Government in the framework of the Flanders AI Research Program (<https://www.flandersairesearch.be/en>) that is financed by EWI (Economie Wetenschap & Innovatie).

REFERENCES

- Adamek, J. and Lucia, S. (2023). Approximate model predictive control based on neural networks in a cloud-based environment. In *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 567–572. IEEE.
- Andersson, J. A., Gillis, J., Horn, G., Rawlings, J. B., and Diehl, M. (2019). Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36.
- Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. (2000). The explicit solution of model predictive control via multiparametric quadratic programming. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 2, pages 872–876 vol.2.
- Biegler, L. T. and Zavala, V. M. (2009). Large-scale nonlinear programming using ipopt: An integrating framework for enterprise-wide dynamic optimization. *Computers & Chemical Engineering*, 33(3):575–582.
- Chen, S., Saulnier, K., Atanasov, N., Lee, D. D., Kumar, V., Pappas, G. J., and Morari, M. (2018). Approximating explicit model predictive control using constrained neural networks. In *2018 Annual American Control Conference (ACC)*, pages 1520–1527.
- Chen, S. W., Wang, T., Atanasov, N., Kumar, V., and Morari, M. (2022). Large scale model predictive control with neural networks and primal active sets. *Automatica*, 135:109947.
- Chen, Y. and Peng, C. (2017). Intelligent adaptive sampling guided by gaussian process inference. *Measurement Science and Technology*, 28(10):105005.
- Docquier, N., Poncelet, A., and Fiset, P. (2013). Robotran: a powerful symbolic generator of multibody models. *Mechanical Sciences*, 4(1):199–219.
- Gupta, S., Paudel, A., Thapa, M., Mulani, S. B., and Walters, R. W. (2023). Optimal sampling-based neural networks for uncertainty quantification and stochastic optimization. *Aerospace Science and Technology*, 133:108109.
- Hertneck, M., Köhler, J., Trimpe, S., and Allgöwer, F. (2018). Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3):543–548.
- Hose, H., Brunzema, P., von Rohr, A., Gräfe, A., Schoellig, A. P., and Trimpe, S. (2024). Fine-tuning of neural network approximate mpc without retraining via bayesian optimization. In *CoRL Workshop on Safe and Robust Robot Learning for Operation in the Real World*.
- Karg, B. and Lucia, S. (2018). Deep learning-based embedded mixed-integer model predictive control. In *2018 European Control Conference (ECC)*, pages 2075–2080.
- Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Kvasnica, M., Löfberg, J., and Fikar, M. (2011). Stabilizing polynomial approximation of explicit mpc. *Automatica*, 47(10):2292–2297.
- Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.
- Lucia, S. and Karg, B. (2018). A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 51(20):511–516. 6th IFAC Conference on Nonlinear Model Predictive Control NMPC 2018.

- Lucia, S., Navarro, D., Karg, B., Sarnago, H., and Lucía, O. (2021). Deep learning-based model predictive control for resonant power converters. *IEEE Transactions on Industrial Informatics*, 17(1):409–420.
- Maddalena, E., da S. Moraes, C., Waltrich, G., and Jones, C. (2020). A neural network architecture to learn explicit mpc controllers from data. *IFAC-PapersOnLine*, 53(2):11362–11367. 21st IFAC World Congress.
- Nubert, J., Köhler, J., Berenz, V., Allgöwer, F., and Trimpe, S. (2020). Safe and fast tracking on a robot manipulator: Robust mpc and neural network control. *IEEE Robotics and Automation Letters*, 5(2):3050–3057.
- Okamoto, M., Ren, J., Mao, Q., Liu, J., and Cao, Y. (2024). Deep learning-based approximation of model predictive control laws using mixture networks. *IEEE Transactions on Automation Science and Engineering*.
- Parisini, T. and Zoppoli, R. (1995). A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10):1443–1451.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- Pozzi, A., Moura, S., and Toti, D. (2022). A neural network-based approximation of model predictive control for a lithium-ion battery with electro-thermal dynamics. In *2022 IEEE 17th International Conference on Control & Automation (ICCA)*, pages 160–165. IEEE.
- Qin, S. J. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764.
- Rawlings, J. B. (2000). Tutorial overview of model predictive control. *IEEE control systems magazine*, 20(3):38–52.
- Singh, T., Mrak, B., and Gillis, J. (2024). Real-time model predictive control for energy-optimal obstacle avoidance in parallel scara robot for a pick and place application. In *2024 IEEE Conference on Control Technology and Applications (CCTA)*, pages 694–701. IEEE.
- Summers, S., Jones, C. N., Lygeros, J., and Morari, M. (2011). A multiresolution approximation method for fast explicit model predictive control. *IEEE Transactions on Automatic Control*, 56(11):2530–2541.
- Vanroye, L., Sathya, A., De Schutter, J., and Decré, W. (2023). Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10036–10043. IEEE.
- Winqvist, R., Venkitaraman, A., and Wahlberg, B. (2021). Learning models of model predictive controllers using gradient data. *IFAC-PapersOnLine*, 54(7):7–12. 19th IFAC Symposium on System Identification SYSID 2021.
- Zhang, X., Bujarbaruah, M., and Borrelli, F. (2019). Safe and near-optimal policy learning for model predictive control using primal-dual neural networks. In *2019 American Control Conference (ACC)*, pages 354–359.