Approximate SDD-TMPC with Spiking Neural Networks
An Application to Wheeled Robots

Surma, F.; Jamshidnejad, A.

DOI
10.1016/j.ifacol.2024.09.050

Publication date
2024

Document Version
Final published version

Published in
IFAC-PapersOnline

Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnline CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 58-18 (2024) 323–328

Approximate SDD-TMPC with Spiking Neural Networks: An Application to Wheeled Robots

F. Surma and A. Jamshidnejad

Control and Operations Department, TU Delft, 2629 HS, Delft, The Netherlands (e-mails: {f.surma;a.jamshidnejad}@tudelft.nl).

Abstract: Model Predictive Control (MPC) optimizes an objective function within a prediction window under constraints. In the presence of bounded disturbances, robust versions are used. Recently, a promising robust MPC was introduced that outperforms SOTA approaches. However, solving the optimization problem online is computationally expensive. An efficient approximation method, such as neural networks (NN), can be substituted to accelerate the online computation. There are discrepancies between the control inputs due to the approximation. We propose to model them as bounded state-dependent disturbances to robustly control nonlinear wheeled robots. We consider a spiking NN to ensure that small robots could use it.

Keywords: Model predictive and optimization-based control, Robust learning systems, Robotics, Nonlinear predictive control, Neural networks.

1. INTRODUCTION

Model predictive control (MPC) is a state-of-the-art control algorithm (Rawlings et al. (2017)) which utilizes a model of the controlled system to estimate optimal control inputs, based on a prediction of the future states of the system (Rawlings et al. (2017)). MPC systematically incorporates the state and input constraints, while it minimizes its objective function within a time window. MPC has been successfully implemented in various systems, including robot control (Chipofya et al. (2015), Sun et al. (2018), Jamshidnejad and Frazzoli (2018)).

Robust versions of MPC ensure that, despite bounded disturbances, the constraints are satisfied. Since robust MPC is designed for the worst-case disturbance scenarios, the performance is compromised. A common robust MPC method is Tube MPC (TMPC), which uses a tube, i.e., a set that includes all possible future states that satisfy the constraints despite the disturbances. Usually, the shape of the tube is designed offline, whereas TMPC determines online the center of the tube, as well as an ancillary control law that keeps the actual states of the system within the tube.

State-dependent dynamic TMPC (SDD-TMPC) was recently introduced by Surma and Jamshidnejad (2023). SDD-TMPC significantly improves the performance and feasibility of the robust control problem, compared to regular TMPC. SDD-TMPC determines a dynamic tube that accounts for the dynamics of the state-dependent disturbances. A main limitation of SDD-TMPC currently is its heavy online computations.

The high online computational complexity that hampers the implementation of Model Predictive Control (MPC) is a well-known problem in the literature. Two solutions have been proposed: Explicit MPC (EMPC) and Approximate MPC (AMPC). In both approaches, all complex computations are performed offline, which allows the controller to work relatively quickly. In EMPC, the state space is divided into regions, each with its unique control law (Grancharova and Johansen (2012)). However, the number of regions increases rapidly with the number of constraints and the prediction horizon (Karg and Lucia (2020)). Thus making this approach inefficient even without solving optimization problem.

AMPC uses a machine learning model to approximate MPC such as neural networks, radial basis functions, or lattice representation (Karg and Lucia (2020)). For linear MPC, it is possible to achieve perfect approximation by using neural networks (NN) with ReLU activation functions. Although linear MPC with a quadratic solver is the fastest feasible implementation of MPC, the speed of computation could still be increased by more than 100 times by using AMPC (Quan and Chung (2019)). To date, AMPC has been used to control, among other things, an inverted pendulum on a trolley (the network had 10 layers with 6 neurons per layer) (Karg and Lucia (2020)), drone (2 hidden layers with 32 neurons per layer) (Tagliabue et al. (2022)) and robotic arm (20 hidden layers with up to 1024 neurons per layer) (Nubert et al. (2020)). The complexity of the MPC law to be approximated determines the required size of the network.

Although neural networks allow for accelerated computation, approximating complex control laws can potentially lead to very deep networks. As deep neural networks can be energy inefficient, it can be challenging to implement them to control small robots such as micro drones (Stroobants

⋆ This research has been supported by the NWO Talent Program Veni project "Autonomous drones flocking for search-and-rescue" (18120), which has been financed by the Netherlands Organisation for Scientific Research (NWO).

10.1016/j.ifacol.2024.09.050

et al. (2023)). In (Henkes et al. (2022)) it has been shown that by using spiking neural networks (SNN) instead, it is possible to reduce power consumption by over 200 times. Although SNNs are rarely used for regression, they have been used as controllers. Examples of implementations include an optic flow-based controller (Dupeyroux et al. (2021)), a deep reinforcement learning trained controller (Tang et al. (2021)), a PID mimicking controller (Burgers et al. (2023)), and an MPC mimicking controller (Halaly and Ezra Tsur (2023)).

In this paper, we will use SNN to approximate SDD-TMPC (approximate SDD-TMPC), enabling the implementation of complex control laws in a fast and energy-efficient manner.

The main contributions of this paper include:

- Introducing a computationally efficient approximate version of SDD-TMPC for wheeled robots, via spiking NNs, also noting that this is the first time that a robust MPC approach is approximated via spiking NNs.
- Proposing an algorithm for efficiently generating the dynamic tube of SDD-TMPC online, taking into account the robot's shape.
- Training an approximate SDD-TMPC and implementing it to control a wheeled robot in numerical simulations. The approximate SDD-TMPC remains safe because the differences between the controller and its approximation are treated as disturbances. Even though large bounds on the disturbances are considered, the controller is not overly conservative because robustness is designed based on state-dependent disturbances, not based on the bounds.

2. APPROXIMATED SDD-TMPC

2.1 Problem formulation - controlling wheeled robot

In (Sun et al. (2018)), a prediction model of a unicycle robot was developed, which is given by (1). The position $(p_{x,t}, p_{y,t})$, in two dimensions and rotation ψ_t of the robot's front head are controlled by velocity ν_t and ω_t. The head of the robot is its front point, located on the main axis of the robot that is perpendicular to the axis of the wheels of the robot. The control objective is to reach a destination without collisions.

$$\dot{\boldsymbol{x}}_t = \begin{bmatrix} \dot{p}_{x,t} \\ \dot{p}_{y,t} \\ \dot{\psi}_t \end{bmatrix} = \begin{bmatrix} \nu_t \cos(\psi_t) + \rho\omega_t \sin(\psi_t) \\ \nu_t \sin(\psi_t) + \rho\omega_t \cos(\psi_t) \\ \omega_t \end{bmatrix} \tag{1}$$

2.2 SDD-TMPC

In Surma and Jamshidnejad (2023), SDD-TMPC solved a closely related obstacle-free problem. SDD-TMPC finds an optimal trajectory of nominal inputs, $\tilde{v}_k$, nominal states $\tilde{z}_k$, and the tube $\mathbb{T}_k$ by solving (2) which is in general a nonlinear, non-convex optimisation problem (2) and may be solved with particle swarm optimisation (Kennedy and Eberhart (1995)). The controller takes as input the real system state x_k and the nominal state z_k. A quadratic cost function is minimized (2a). A discretized state equation $f^d(\cdot)$ is used (2b) to predict future nominal states. Since it is not possible to predict the real state perfectly, we use a set $\mathbb{E}_k$, which contains all the possible errors between the nominal and real states. The evolution of the error set is described by (2c). An error is a difference between the actual state and the nominal state. Since the future actual state is unknown, the error set contains all possible future error sets. Although there is no state-dependent external disturbance, a model of the error dynamics of the direction has been simplified. To account for this, a difference between the real and the simplified model is treated as a state-dependent disturbance $w_t \in \mathbb{W}(\boldsymbol{x}_t)$. The first error set is just a difference between real and nominal state (2d). The original control problem contained only the input constraints (2e) and the terminal state constraints (2f). After solving the optimization problem, the real controller input is computed with ancillary control law (3) and depends on the real state, the nominal state, and the nominal input. In the next iteration, the nominal state is equal to the state that the robot would reach from the nominal state if the nominal input was applied and the disturbance was equal to 0. SDD-TMPC ensures recursive feasibility and stability. We have:

$$V^*(\boldsymbol{x}_k, \boldsymbol{z}_k) = \min_{\tilde{\boldsymbol{z}}_k, \tilde{\boldsymbol{v}}_k, \mathbb{T}_k} V(\boldsymbol{x}_k, \tilde{\boldsymbol{z}}_k, \tilde{\boldsymbol{v}}_k, \mathbb{T}_k) = \tag{2a}$$

$$= \min_{\tilde{\boldsymbol{z}}_k, \tilde{\boldsymbol{v}}_k, \mathbb{T}_k} \sum_{i=k}^{k+N-1} \left(\left\| \boldsymbol{z}_{i|k} \right\|_Q^2 + \left\| \boldsymbol{v}_{i|k} \right\|_R^2 \right) + \left\| \boldsymbol{z}_{k+N|k} \right\|_F^2$$

$$\text{s.t. for } i = k+1, \ldots, k+N:$$

$$\boldsymbol{z}_{i+1|k} = f^{\mathrm{d}}(\boldsymbol{z}_{i|k}, \boldsymbol{v}_{i|k}) \tag{2b}$$

$$\bar{\mathbb{E}}_{i|k} = \operatorname{diag}\left(K^{\mathrm{e,d}}, K^{\mathrm{e,d}}, K^{\theta,\mathrm{d}}\right) \mathbb{E}_{i-1|k} \oplus \mathbb{W}_{i-1|k}(\cdot) \tag{2c}$$

$$\mathbb{E}_k = \{\boldsymbol{x}_k - \boldsymbol{z}_k\} \tag{2d}$$

$$v_{i|k} \in \mathbb{V}(\mathbb{E}_{i|k}) \tag{2e}$$

$$\boldsymbol{z}_{N|k} \in \mathbb{Z}_{\mathrm{f}} \tag{2f}$$

Then the control input (including an online ancillary input) is determined after discretization of:

$$\boldsymbol{u}_t = \begin{bmatrix} \cos(\boldsymbol{x}_t[3]) & -\rho\sin(\boldsymbol{x}_t[3]) \\ \sin(\boldsymbol{x}_t[3]) & \rho\cos(\boldsymbol{x}_t[3]) \end{bmatrix}^{-1} \tag{3}$$

$$\left(\begin{bmatrix} \cos(\boldsymbol{z}_t[3]) & -\rho\sin(\boldsymbol{z}_t[3]) \\ \sin(\boldsymbol{z}_t[3]) & \rho\cos(\boldsymbol{z}_t[3]) \end{bmatrix} \boldsymbol{v}_t - K^{\mathrm{e}} \boldsymbol{e}_t[1:2] \right)$$

In this paper SDD-TMPC is extended. If the original SDD-TMPC were replaced, the network would need the nominal state as an input, but it would not be possible to compute it, as the output of the network would only contain the input, because it is not possible to generate a nominal control with a network, as it would add disturbances to the nominal state, but by definition the nominal state cannot be affected by disturbances. Otherwise, the stability proofs would not hold. To solve this problem, we use additional constraint (4a) to initialize the first nominal state based only on the current state and the first nominal state is now a decision variable. In such a case, the difference between the nominal state and the actual state must be within the largest possible tube $\mathbb{E}^{\max}$, which can be found via the procedure that is given by (Sun et al. (2018)).

Since there are obstacles in the environment, state constraints on the position are added (4b). However, since

the robot has its shape, it must be ensured that not only the head of the robot avoids collisions, but the entire body. The shape function returns the tube consisting of all possible shapes of the whole body. Many wheeled robots such as e-pucks, turtlebots, or irobots can be represented as a circle.

In this case, the tube of all shapes can be represented by a polytope. This polytope can be constructed in the following steps:

- Choose a set of directions (e.g. $\{0, \pi/2, \pi, 3\pi/2\}$). The more directions are chosen, the more optimal will be the final solution, but the computation cost for solving the optimization problem will be increased.
- For each direction θ_k compute the distance with (4c), where r is a radius of a robot and $e_k[3]$ is a third element of the vector e_k, i.e. a possible error in the direction.
- Construct a polytope, where each equation defining the polytope can be written as (4d). An example of such a polytope can be seen in the figure 1.
- To the set obtained in the previous point, add (using Minkowski summation, i.e. the output of this operation is a set containing all possible sums of all possible pairs of elements from both input sets) a box set containing all possible positional errors.

$$\boldsymbol{x}_k \in \{\boldsymbol{z}_k\} \oplus \mathbb{E}^{\max} \tag{4a}$$

$$\text{shape}(z_{i|k} \oplus \mathbb{E}) \in \mathbb{X} \tag{4b}$$

$$\text{dist}_i = \rho(\cos(\min_{e \in \mathbb{E}}(|e[3] - \theta_i|)) + 1) \tag{4c}$$

$$-\cos(\theta_i)p_x - \sin(\theta_i)p_y \leq \text{dist}_i \tag{4d}$$

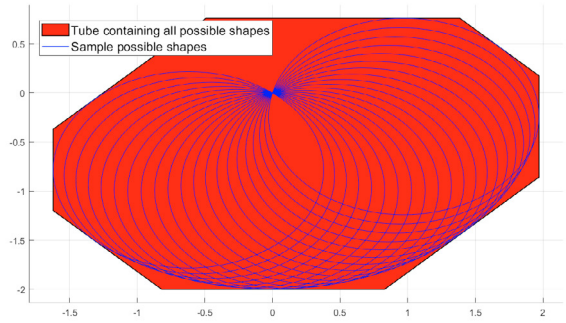

Fig. 1. An example showing the possible shapes of a robot when the robot's directions are between 0.9 and 2.9 rad, and the tube contains all possible shapes.

2.3 Approximating MPC with SNN

In classical neural network architecture, a neuron takes a set of inputs. These inputs can come from the environment or other neurons in the network. The inputs are multiplied by weights found in the training process. The resulting values are summed and passed through a nonlinear activation function. The final value is an output from the neuron. In a feedforward network, the neurons are grouped into layers. The outputs of the neurons in one layer are sent as inputs to the next layers.

In a regression task, the output of the final layers must be a physical value. The network can be trained using a back-propagation algorithm. It uses a mean square error between the outputs of the network and the real data. The new weights are computed using the stochastic gradient descent algorithm (Bottou (2010)).

Since all neurons send information all the time, it can lead to inefficient energy consumption compared to SNN (Eshraghian et al. (2023)). In SNN, each neuron has a memory value m. In the most common application of leak-and-fire (LiF) neurons, the dynamics of a neuron can be described by (5), where β is a constant hyperparameter, w_i is the weight between the current neuron and its $i-th$ input, σ_i is an $i-th$ input (this value is 0 or 1 if it is also the output of another neuron), and a reset value that decreases the memory value when the neuron fires a spike.

$$m_{k+1} = \beta m_k + \sum_{i=1}^{n}(w_i\sigma_i) - \text{reset}(m_k) \tag{5}$$

Our network is based on Henkes et al. (2022), where the first input of the network is not a spike but a continuous value symbolizing the real input, i.e. the state of the robot. Then all neurons in the hidden layers are LiF neurons. However, in the final output layer, the neurons no longer fire spikes, and we treat the memory value of each of them as the output of the network. One could argue that this is no longer an SNN, but this is not important as this architecture still allows energy to be saved.

To collect data for training the network, we simulate a randomly sampled state to compute and store a control input for that state. This process is repeated until a sufficiently large data set is generated. It is important to note that the solver may find a local minimum that is not necessarily close to the global minimum, so we spend more time (one hour per sample) searching for the solution. The reason for this is that local minima can be treated as outliers during SNN training, what makes the training process more difficult.

There are several problems with the use of SNN. First, the reset mechanism is a step function and cannot be differentiated. Its gradient is 0 or infinite. When stochastic gradient descent is used, the derivative of the step is replaced during backpropagation by the derivative of a function called the surrogate gradient. This makes it more difficult to train SNN, but there is no other choice if you want to use gradient-based approaches.

Second, if MPC has access to the current state, it can find the control input, i.e., it does not need past information. SNN, on the other hand, requires memory to operate. However, SNN has been shown to work even with a time-invariant dataset such as the MNIST dataset (Subbulakshmi Radhakrishnan et al. (2021)). This can be done by simulating the network for some constant time steps and taking the final values of the memory as the output of the network after those time steps. However, in such a case it is necessary to use the backpropagation through time algorithm instead of the standard backpropagation, which takes more real time to train.

Third, in our experience, if the initial weights are generated randomly, it is likely that some neurons will not fire at all, no matter what the input is. This makes their gradient very small, which means that their weights are rarely updated during training. Such neurons can be called dead neurons (Eshraghian et al. (2023)). In training, we use mean square error, but to solve this problem we add an additive regularization term that penalizes the network if some neurons fire less than a certain threshold for the whole batch, i.e. it is fine if a neuron does not fire for a certain state, but if it does not fire at all, it is penalized.

2.4 Constraint satisfaction

The main advantage of robust MPC is in providing safety, i.e., in ensuring that none of the constraints is ever violated. To ensure constraint satisfaction after replacing SDD-TMPC with its approximate SNN, we consider a disturbance $\boldsymbol{d}_k$ because the MPC control input $\boldsymbol{u}_k^{\text{MPC}}$ is not perfectly mimicked by the SNN input $\boldsymbol{u}_k^{\text{SNN}}$, as indicated in (6). We assume $\boldsymbol{d}_k$ is bounded i.e. $\boldsymbol{d}_k \in \mathbb{D}$. By adding $\boldsymbol{d}_k$ to the original model, (1) results in (7). This leads to a direction-dependent set $\mathbb{W}^d_{i-1|k}$ that is obtained via a nonlinear mapping of the original boundary set for the disturbances, as it is given in (8). We then include the knowledge of this perturbation within SDD-TMPC by replacing $\mathbb{W}_{i-1|k}(\cdot)$ in (2c) with $\mathbb{W}_{i-1|k}(\cdot) \oplus \mathbb{W}^d_{i-1|k}(\cdot)$. Even though we are working with input discrepancy, since this has been modeled as a state-dependent additive disturbance, SDD-TMPC guarantees the satisfaction of the state constraints.

As long as the assumption $\boldsymbol{d}_k \in \mathbb{D}$ holds, the constraint will never be violated, and it is possible to compute the probability of the assumption being satisfied using Hoeffding's inequality (Hertneck et al. (2018)), but the drawback of this approach is that it requires a lot of samples.

It may be impossible to estimate the set $\mathbb{D}$ before training the network, but if the results are not satisfactory, generating more data and training a more complex network is possible.

$$\boldsymbol{d}_k = [d_k^{\nu}\ d_k^{\omega}]^T = \boldsymbol{u}_k^{\text{MPC}} - \boldsymbol{u}_k^{\text{SNN}} \tag{6}$$

$$\dot{\boldsymbol{x}}_t = \begin{bmatrix} \cos(\psi_t) & \rho\sin(\psi_t) \\ \sin(\psi_t) & \rho\cos(\psi_t) \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \nu_t + d_t^{\nu} \\ \omega_t + d_t^{\omega} \end{bmatrix} \tag{7}$$

$$\mathbb{W}^d(\psi) = \begin{bmatrix} \cos(\psi_t) & \rho\sin(\psi_t) \\ \sin(\psi_t) & \rho\cos(\psi_t) \\ 0 & 1 \end{bmatrix} \mathbb{D} \tag{8}$$

3. SIMULATION RESULTS

In this paper, we have simulated an irobot create3. All code we have used can be found in the published repository (Surma (2024)), which also contains a ROS2 package. We approximated the shape of this robot by a circle with a radius r equal to 23.7 cm. The maximum linear velocity $\nu^{\max}$ is equal to 0.46 $\frac{\text{m}}{\text{s}}$ and the input of the robot is constrained by $\nu + r\omega \leq \nu^{\max}$.

At each iteration, the robot was randomly placed in a circular area with a radius of 1.08 m surrounded by a wall. A robot was instructed to minimize its position and its inputs with cost matrices equal to Q=diag(0.4,0.4), R=diag(0.2,0.2), and F = diag(0.5,0.5). The horizon was set to 20. We assumed that the disturbance $\boldsymbol{d}$ is at most 20% of the maximum input value. For this disturbance, we can define the largest possible error set $\mathbb{E}_{\max} =: \{\boldsymbol{e} \in \mathbb{R}^3 | \max(|\boldsymbol{e}[1]|, |\boldsymbol{e}[2]|) \leq 0.1\text{m}, |\boldsymbol{e}[3]| \leq \pi\}$. The robot's control trajectory had to satisfy the control input constraint, never let the robot collide with a wall, and let the nominal state reach the terminal set, which is a circle with a radius equal to 0.5 m.

In the next step, we used the collected data to train 2 SNNs: Linear Velocity Network (LNV) and Angular Velocity Network (AVN). We did this because learning the rotational velocity proved to be more challenging. As a result, a single SNN was more focused on minimizing the angular velocity, which led to a higher number of outliers while trying to predict the linear velocity. We created 1168 data samples (900 for training and 268 for validation). We were able to ignore the direction by always rotating the coordinate system by robot direction before generating the data. The same approach was used in (Tagliabue et al. (May 23-27, 2022)). The LNV has 5 hidden layers with 50, 200, 1000, 200, and 50 neurons in the layers and the network was simulated for 20 iterations per control input. The rotation control input proved to be more difficult and AVN had 7 hidden layers with 150, 200, 1000, 1200, 1000, 500, and 150 and was simulated for 40 iterations. In both networks, we used Atan function as a surrogate gradient and hard reset (after spiking a neuron's memory was set to 0).

From our experience, longer simulations of SNN work as a regularization factor, i.e. it reduces the validation error, but it makes training much more time-consuming. Thanks to this regularization factor, it was possible to train a deeper neural network with less data. We trained the network using the Adam solver with a learning rate 10^{-4} and a batch size of 32.

According to our training for LVN, the average training and validation errors are 0.0326 and 0.0413 respectively. All training errors shown are relative to the maximum control input value. The highest error is equal to 0.156, which means that our initial assumption is very likely to be satisfied. According to our training for AVN, the average training and validation errors are 0.0373 and 0.0431 respectively. The highest error is 0.233 and the assumption $\boldsymbol{d}_k \in \mathbb{D}$ was not satisfied 4 times (3 times in training and 1 time in the validation dataset). It is possible to solve this problem by creating a larger set of possible disturbances, generating new data, and retraining the network. However, even though the initial assumption is not always satisfied, we learned during simulations that if it happens sporadically it does not affect the performance too much, but increasing the disturbance set would occur in a more overly conservative controller.

In the last step, we compared both ASDD-TMPC and SDDTMPC. In figure 2 we can see the consistent behavior of the controller when the robot has to move from position [0.5,0.5] and orientation 0.5π to the origin. It shows the

position of the head over time using both controllers. To better demonstrate what the robot is doing (i.e. it first corrected its orientation by moving backward, then it turned to reach the target), the line between the center and the head showing the orientation was added. We can see that the behavior of both controllers is very similar, but ASDD-TMPC was slightly more aggressive.

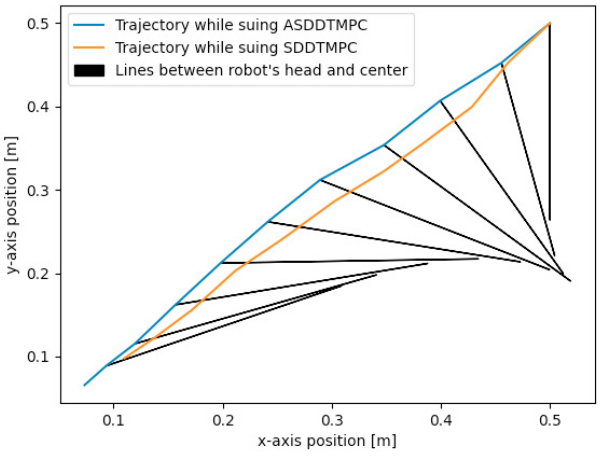

Fig. 2. The transient behavior of the robot's head while controlled by two controllers. The black lines show the distance between the center and the head to better explain the robot's behavior (ASDD-TMPC only).

An important challenge is presented by Figure 3, where SDD-TMPC can reach the target, but by using ASDD-TMPC a few centimeters of steady-state error is displayed. This is what we should expect, because according to the largest tube, the largest steady-state error could even be over 0.15 cm, but it is unlikely since the average approximation error is usually much lower than the largest disturbance according to the assumption. One way to solve this problem would be to use a different controller at the end. Figure 4 shows how the distance changes and even though ASDD-TMPC is more aggressive, the discrepancy in the control input leads to the constant steady-state error. Finally, Figure 5 shows that the relationship between the inputs of both controllers is very similar, with ASDD-TMPC again being more aggressive, but it also clearly shows how ASDD-TMPC is affected by the input disturbance.

4. CONCLUSION

In this paper, we extended the original SDD-TMPC where the main drawback was the time required to compute a control input. This problem was solved by approximating this controller using spiking neural networks for the first time, which reduced the online computation time but introduced an additional control input. Based on our results, the time required to compute the control input was reduced from minutes to milliseconds, a massive improvement. In addition, we extended the controller to take into account the shape of the robot, even when the direction is uncertain.

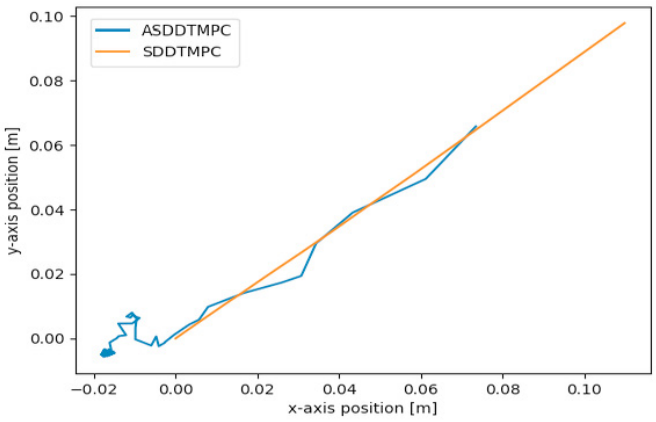

Fig. 3. The stable behavior of the robot's head while controlled by two controllers.

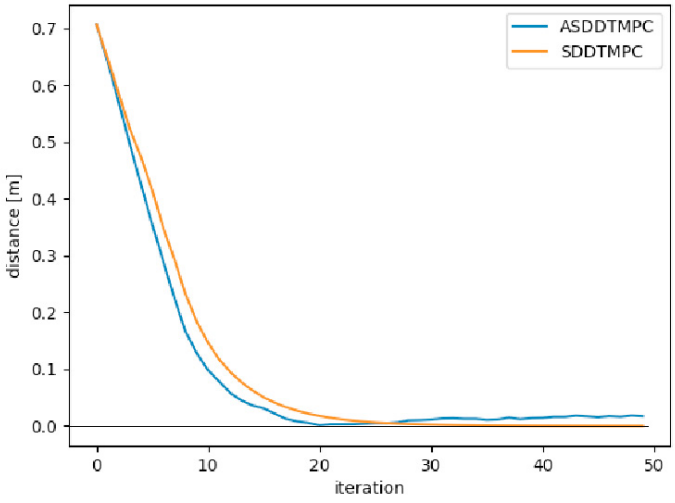

Fig. 4. Distance between robot head and origin over time

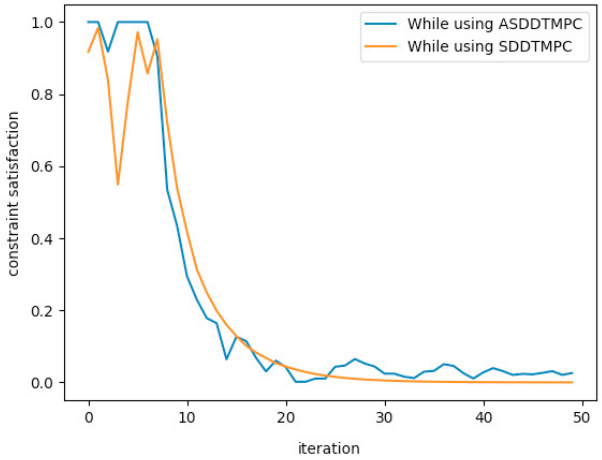

Fig. 5. Control input over time. It is scaled so that if the value is 1, the robot cannot move faster with the same ratio between linear and angular velocity.

According to our dataset, two trained networks were able to imitate the original controller, rarely breaking the original assumption that the disturbance in the control input is less than a chosen constant. However, if safety is

critical, it is always possible to retrain the network using SDD-TMPC with a larger disturbance set. While training the neural network, we did not make any assumptions about the model, cost function, etc., so this approach applies to other control problems as long as there is enough data and the network has a suitable architecture (i.e., number of layers and neurons)

The main challenge for now is the generalization, i.e. every time the model or the constraints change (e.g. the robot works in a different environment), the networks have to be re-trained, which makes it impossible to control the robot in an unknown environment. The next critical step would be to add information gathered by sensors as another input of the network. Another important extension would be to find a way to generate samples more efficiently, as currently it takes a lot of time and many more samples to mimic SDD-TMPC for more complex systems.

ACKNOWLEDGEMENT

This research has been supported by the NWO Talent Program Veni project "Autonomous drones flocking for search-and-rescue" (18120), which has been financed by the Netherlands Organisation for Scientific Research (NWO).

REFERENCES

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta (eds.), *Proceedings of COMPSTAT'2010*, 177–186. Physica-Verlag HD, Heidelberg.

Burgers, T., Stroobants, S., and de Croon, G. (2023). Evolving spiking neural networks to mimic pid control for autonomous blimps.

Chipofya, M., Lee, D., and Chong, K. (2015). Trajectory tracking and stabilization of a quadrotor using model predictive control of laguerre functions. *Abstract and Applied Analysis*, 2015. doi:10.1155/2015/916864.

Dupeyroux, J., Hagenaars, J.J., Paredes-Vallés, F., and de Croon, G.C.H.E. (2021). Neuromorphic control for optic-flow-based landing of mavs using the loihi processor. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 96–102.

Eshraghian, J.K., Ward, M., Neftci, E.O., Wang, X., Lenz, G., Dwivedi, G., Bennamoun, M., Jeong, D.S., and Lu, W.D. (2023). Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, 111(9), 1016–1054. doi:10.1109/JPROC.2023.3308088.

Grancharova, A. and Johansen, T. (2012). *Explicit Nonlinear Model Predictive Control: Theory and Applications*, volume 429. doi:10.1007/978-3-642-28780-0.

Halaly, R. and Ezra Tsur, E. (2023). Autonomous driving controllers with neuromorphic spiking neural networks. *Frontiers in Neurorobotics*, 17.

Henkes, A., Eshraghian, J., and Wessels, H. (2022). Spiking neural networks for nonlinear regression. doi:10.48550/arXiv.2210.03515.

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. doi:10.1109/LCSYS.2018.2843682.

Jamshidnejad, A. and Frazzoli, E. (2018). Adaptive optimal receding-horizon robot navigation via short-term policy development. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 21–28. IEEE.

Karg, B. and Lucia, S. (2020). Efficient representation and approximation of model predictive control laws via deep learning. *IEEE Transactions on Cybernetics*, 50(9), 3866–3878. doi:10.1109/TCYB.2020.2999556.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, 1942–1948 vol.4. doi:10.1109/ICNN.1995.488968.

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*, PP, 1–1. doi:10.1109/LRA.2020.2975727.

Quan, Y.S. and Chung, C.C. (2019). Approximate model predictive control with recurrent neural network for autonomous driving vehicles. In *2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 1076–1081. doi:10.23919/SICE.2019.8859955.

Rawlings, J., Mayne, D., and Diehl, M. (2017). *Model Predictive Control: Theory, Computation, and Design 2nd edition*. Nob Hill Publishing. Madison, Wisconsin, USA.

Stroobants, S., De Wagter, C., and De Croon, G. (2023). Neuromorphic control using input-weighted threshold adaptation. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, ICONS '23. Association for Computing Machinery, New York, NY, USA. doi:10.1145/3589737.3605963.

Subbulakshmi Radhakrishnan, S., Sebastian, A., Oberoi, A., Das, S., and Das, S. (2021). A biomimetic neural encoder for spiking neural network. *Nature Communications*, 12. doi:10.1038/s41467-021-22332-8.

Sun, Z., Dai, L., Liu, K., Xia, Y., and Johansson, K.H. (2018). Robust mpc for tracking constrained unicycle robots with additive disturbances. *Automatica*, 90, 172–184.

Surma, F. (2024). *Implemenation of Appoximate state-dependent dynamic TMPC*. doi:10.4121/6568e235-289d-4be0-b931-ec219c559f6f.

Surma, F. and Jamshidnejad, A. (2023). State-dependent dynamic tube mpc: A novel tube mpc method with a fuzzy model of disturbances.

Tagliabue, A., Kim, D.K., Everett, M., and How, J. (2022). Demonstration-efficient guided policy search via imitation of robust tube mpc. 462–468. doi:10.1109/ICRA46639.2022.9812122.

Tagliabue, A., Kim, D.K., Everett, M., and How, J.P. (May 23-27, 2022). Demonstration-efficient guided policy search via imitation of robust tube MPC. In *International Conference on Robotics and Automation*, 462–468. doi:10.1109/ICRA46639.2022.9812122. Philadelphia, Pennsylvania, USA.

Tang, G., Kumar, N., Yoo, R., and Michmizos, K. (2021). Deep reinforcement learning with population-coded spiking neural network for continuous control. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155 of *Proceedings of Machine Learning Research*, 2016–2029. PMLR.