Model Predictive Control for Quadrupedal Robots with Neural-based Adaptation

Dmitry Bazylev[1], Maxim Lyahovski[1] and Dmitrii Dobriborsci[2]

Abstract— **This paper addresses the problem of a synthesis of model predictive control for quadrupedal robots with adaptive change of weight matrices using a neural network. The robot model in the predictive control algorithm is considered as a single rigid body, which is affected by forces in the contact spots. The control method is a combination of a swing leg and ground force controllers with the use of whole-body impulse control for the latter. A fully connected neural network is applied for automatic tuning of weight coefficients used in convex model predictive control. The effectiveness of the proposed approach is demonstrated using numerical simulations, which provide a significant reduction in errors for various robot speeds.**

I. INTRODUCTION

Quadruped robots have attracted considerable attention in recent years due to their ability to traverse complex terrain and perform dynamic maneuvers. Unlike wheeled robots, quadrupeds offer superior mobility in unstructured environments such as disaster zones and natural landscapes. However, controlling these robots presents significant challenges, including maintaining stability, adapting to changing ground conditions, and performing flexible movements under insufficient actuation and dynamic constraints [1].

Practical applications of quadruped robots include monitoring the condition and ensuring safety measures at factories [2], mapping cave systems, creating virtual doubles of rooms, accompanying people with disabilities, searching and rescuing people in emergency situations, etc. [3].

The control of quadrupedal robots encompasses multiple levels of decision-making, from high-level path planning to low-level joint torque control. Key approaches include model-based optimization techniques, such as Model Predictive Control (MPC) and Zero-Moment Point (ZMP)-based methods, as well as learning-based strategies leveraging reinforcement learning (RL) and neural networks. Hybrid methods that combine classical control theory with data-driven learning have shown promise in improving robustness and adaptability.

Recent advances in reinforcement learning have demonstrated remarkable performance in legged robot control, as evidenced by contemporary research. For instance, [4] presents an RL-based controller trained in simulation within just a few hours using trust region policy optimization [5], which surpasses the performance of state-of-the-art model-based controllers on the same robotic platform. Further extending this paradigm, [6] integrates local navigation and locomotion through an end-to-end deep RL policy.

Despite advantages of RL-based methods, their practical implementation in legged robotics remains constrained by the inevitable gap between modeling and reality that occurs when policies developed during simulations are transferred, the difficulty of designing reward functions for desired robot behavior, and the absence of strict stability guarantees.

A foundational approach in model-based optimal control is the Linear-Quadratic Regulator (LQR), which was applied for bipedal robot stabilization using a variational dynamics model in [7]. While effective, LQR exhibits limitations in handling transient disturbances.

To address this, MPC emerged as a more robust alternative, as demonstrated in [8]. MPC leverages a linearized dynamics model around the robot's nominal state and outperforms LQR in rejecting short-term perturbations due to its receding-horizon optimization. Further refinements introduced heuristic state-dependent adjustments (e.g., coupling desired body pitch to translational velocities) to enhance stabilization during locomotion [9]. These heuristics were systematically derived using a dedicated software framework [10].

A critical aspect of MPC is the proper selection of weight matrices in the cost function, which dictate the trade-off between tracking accuracy, control effort, and robustness. Traditionally, these weights are tuned empirically, a time-consuming and often suboptimal process.

For nonlinear systems, approaches like the Nonlinear Quadratic Regulator (NQR) [11] and Nonlinear MPC (NMPC) [12] offer improved accuracy but incur high computational costs, particularly over long prediction horizons. This trade-off spurred the adoption of hybrid methods, such as combining Whole-Body Impulse Control (WBIC) with MPC [9], [13]. WBIC efficiently handles contact dynamics, while MPC optimizes high-level motion, making this framework particularly suitable for small-scale quadrupeds.

However, despite these advances, reproducing and adapting these methods to different robotic platforms remains a challenge. One important challenge is the numerous hard-coded parameters and heuristics embedded in the algorithms, often without clear guidance on their configuration and tuning. This lack of transparency complicates implementation, especially since MPC performance critically depends on an accurate system model.

Supported by the Ministry of Science and Higher Education of the Russian Federation (project no. FSER-2025-0002).

[1]Faculty of Control Systems and Robotics, ITMO University, 49 Kronverkskiy av., 197101 Saint Petersburg, Russia. bazylevd@itmo.ru, maxim.lyahovsky@gmail.com

[2] The author is with Deggendorf Institute of Technology, Technology Campus Cham, Badstrasse 21, 93413 Cham. The contribution of the author is not supported by the project above. dmitrii.dobriborsci@th-deg.de

Recent advances in machine learning (ML) offer promising solutions to automate and optimize MPC tuning. The papers [14] and [15] present research on combining reinforcement learning and MPC in a single controller. By leveraging data-driven approaches such as reinforcement learning (RL), Bayesian optimization, or gradient-based methods, the weight matrices can be adaptively adjusted. These techniques enable quadrupedal robots to learn optimal control policies from experience, either in simulation or real-world deployment, while maintaining the stability guarantees of MPC.

In this work, we explore the use of fully connected neural networks (FCNNs) to dynamically adjust MPC weight matrices in response to locomotion conditions. Unlike black-box reinforcement learning policies that replace the entire controller, our approach retains the interpretability and stability of MPC while leveraging FCNNs to predict optimal weights. Our results suggest that learning-based weight adaptation can enhance the performance of quadrupedal robots while maintaining the reliability of traditional MPC.

Let us introduce the following basic notations that are applied throughout this paper.

Notation 1.

1 $\mathbb{R}$ and $\mathbb{R}_+$ are the sets of real and real positive numbers, respectively. $\mathbb{R}^n$ and $\mathbb{R}^{n\times n}$ are the n and $n\times n$ dimensional Euclidean spaces with the vector norm $\|\cdot\|$.

2 $\overline{1,m}$ denotes a sequence of integer numbers $1,...,m$.

3 $\operatorname{diag}\{\lambda_i\}_{i=1}^n$ is the diagonal matrix with elements λ_i, $i=\overline{1,n}$.

The remaining of the paper is organized as follows. The simplified robot model and problem formulation are given in Section II. Section III presents the main result of the paper with the key steps of synthesis of MPC control technique and application of FCNN for adaptation of weight matrices used in convex MPC. Representative simulation results that demonstrate the efficiency of the proposed approach are given in Section IV. The paper is wrapped-up with concluding remarks in Section V.

II. ROBOT MODEL AND PROBLEM FORMULATION

In this paper we consider the *simplified* robot model with a single rigid body dynamics neglecting the leg dynamics. Such model is frequently used for control synthesis of quadruped robots since that is reasoned by low computational resources required to generate control signals. Thus, the prediction horizon can be updated frequently ensuring small delays in control channel.

Assumption 1: The mass of the robot's legs $m_l \in \mathbb{R}_+$ is significantly lower than the total robot's mass $m \in \mathbb{R}_+$: $m_l << m$.

Remark 1. Assumption A1 is applicable for a number of quadrupedal robots. For instance, the ratio $m_l/m \approx 10\%$ for such robots as MIT Cheetah 3 and Unitree A1.

According to Newton's second law the dynamics of the solid body robot in the global (world) frame is given by

$$\ddot{p} = \frac{\sum_{i=1}^n f_i}{m} - g, \tag{1}$$

where $p \in \mathbb{R}^3$ is the vector of robot's position in the three-dimensional space, $m \in \mathbb{R}_+$ is the robot's body mass, $f_i \in \mathbb{R}^3, i=\overline{1,n}$ is the vector of the reaction forces on the i-th leg, $g \in \mathbb{R}^3$ is the vector of gravitational acceleration with respect to the global frame, and n is the number of contact spots (points).

The behavior of a rotating body is presented by Euler's equation of motion and non-stationary transformation from body to the global frame

$$\begin{aligned} \frac{d}{dt}(I\omega) &= I\dot{\omega} + \omega\times(I\omega) = \sum_{i=1}^n r_i \times f_i, && (2)\\ \dot{R}(\phi,\theta,\psi) &= [\omega]\cdot R(\phi,\theta,\psi), && (3)\end{aligned}$$

where $\omega \in \mathbb{R}^3$ is the vector of angular velocities, $I \in \mathbb{R}^{3\times 3}$ is the inertia matrix of the robot's body, $r_i \in \mathbb{R}^3$ is the vector directed from the center of mass (CoM) of the robot's body to the i-th contact patch, $\phi \in \mathbb{R}$, $\theta \in \mathbb{R}$ and $\psi \in \mathbb{R}$ represent roll, pitch and yaw, respectively, and $R(\phi,\theta,\psi) \in \mathbb{R}^{3\times 3}$ is the rotation matrix which performs transformation from body to global coordinates.

Remark 2 The vector of ground reaction forces in equations (1)-(2) is generated using MPC algorithm which is given in Section III. The length $r_{ij}, i=\overline{1,n}, j=\overline{1,3}$ from equation (2) is the equivalent of the contact force moment arm. In (3) operation $[\cdot]$ applied to ω means take a skew-symmetry matrix from vector $a\omega$.

Notice, that the part of model dynamics presented by (1)-(2) is nonlinear that is reasoned by the dynamics of robot orientation and cross-product components. Therefore, linear MPC cannot be applied at this step.

Following [8], we introduce three reasonable assumptions for the robot model that result in valuable simplification and approximation.

Assumption 2: The roll $\phi(t)$ and pitch $\theta(t)$ angles are sufficiently small: $\phi(t) \sim 0$, $\theta(t) \sim 0$.

Assumption 3: The pitch and roll velocities, $\dot{\phi}(t)$ and $\dot{\theta}(t)$, respectively, as well as off-diagonal components of the inertia matrix I take sufficiently small values: $\dot{\phi}(t) \sim 0$, $\dot{\theta}(t) \sim 0$.

Assumption 4: The current robot's state variables are close enough to the desired (reference) ones.

Application of the Assumption 2 results in simplification of the coordinate transformation for the global rotational velocities $\dot{\Theta} = \begin{bmatrix}\dot{\phi} & \dot{\theta} & \dot{\psi}\end{bmatrix}^\top$

$$\dot{\Theta} = R_z(\psi)\omega. \tag{4}$$

The Assumption 3 allows to perform a valuable approximation of the equation (2), obtaining

$$\frac{d}{dt}(I\omega) = I\dot{\omega} + \omega\times(I\omega) \approx I\dot{\omega}, \tag{5}$$

where the inertia matrix in the global coordinate system can be calculated from the local (body) one I_B as follows

$$\hat{I} = R_z(\psi) I_B R_z^\top(\psi).$$

Finally, under the Assumption 4, we derive a time-varying linear approximation of the system dynamics by

i) incorporating desired trajectories through the rotational transformation matrix $R_z(\psi)$, and ii) computing the moment arm in equation (2) using predetermined values from both the desired trajectory and step locations.

As a result, the continuous dynamics of the robot can be represented in the input-state-output form

$$\frac{d}{dt}\begin{bmatrix}\hat{\Theta}\\ \hat{p}\\ \hat{\omega}\\ \hat{\dot{p}}\end{bmatrix} = \begin{bmatrix}0_3 & 0_3 & R_z^T(\psi) & 0_3\\ 0_3 & 0_3 & 0_3 & 1_3\\ 0_3 & 0_3 & 0_3 & 0_3\\ 0_3 & 0_3 & 0_3 & 0_3\end{bmatrix}\cdot\begin{bmatrix}\hat{\Theta}\\ \hat{p}\\ \hat{\omega}\\ \hat{\dot{p}}\end{bmatrix} + \begin{bmatrix}0_3 & \dots & 0_3\\ 0_3 & \dots & 0_3\\ \hat{I}^{-1}[r_1] & \dots & \hat{I}^{-1}[r_n]\\ 1_3/m & \dots & 1_3/m\end{bmatrix}\cdot\begin{bmatrix}f_1\\ \vdots\\ f_n\end{bmatrix} + \begin{bmatrix}0\\0\\0\\g\end{bmatrix}, \tag{6}$$

where $0_3, 1_3, \hat{\Theta}$ are correspondingly zero value and identity matrix 3×3 and the robot body global rotational angle. To bring the model to a linear form, we add one more condition to the 12 states: $\dot{x}_{13} = 0$; $x_{13}(0) = g$. Thus, we get

$$\dot{x}(t) = A_c(\psi)x(t) + B_c(r_1;\dots r_n;\psi)u(t), \tag{7}$$

where $A_c(\psi) \in \mathbb{R}^{13\times13}$ and $B_c(r_1;\dots r_n;\psi) \in \mathbb{R}^{13\times3n}$.

Finally, we obtain the discrete dynamics using a zero order hold discrete time calculation of $A_c(\psi)$ and $B_c(r_1;\dots r_n;\psi)$ in the form

$$x(k+1) = A_k x(k) + B_k u(k). \tag{8}$$

The objective is to design robot model control based on MPC approach under the given Assumptions 1–4 and perform automatic optimization of weight matrix components used in MPC applying FCNNs reducing tracking errors.

III. MAIN RESULT

First, we consider MPC problem for the quadruped robot. Discrete-time finite-horizon model predictive control is formulated in a standard form and applied to calculate the desired ground reaction forces without requiring knowledge of the leg's configuration or kinematics. Next, we provide the key steps of synthesis of the FCNN used to automatically adjust the weight coefficients in MPC for various robot operation scenarios.

At each control iteration, the MPC controller begins from the current state and optimizes the sequence of control inputs along with the corresponding state trajectory over a finite prediction horizon. Such optimization is repeated every iteration and only the first control input from the computed trajectory is executed before the controller re-evaluates and generates new inputs. Given a discrete-time system with state $x(k)$, control input $u(k)$, and horizon K, MPC solves the following optimization problem at each timestep:

$$\min_{x,u_0,\dots,u_{K-1}} \sum_{k=0}^{K-1}\left(\|x(k+1) - x_{\text{ref}}(k+1)\|_Q^2 + \|u(k)\|_R^2\right) \tag{9}$$

$$\text{subject to}\quad x(k+1) = A_k x(k) + B_k u(k) \tag{10}$$

$$u(k) \in \mathscr{U}, \quad x(k) \in \mathscr{X}, \tag{11}$$

where $Q \succeq 0$ and $R \succeq 0$ are the weight (cost) matrices for the state error and input penalty, respectively, and input $\mathscr{U}$ and state $\mathscr{X}$ constraints are given by

$$\mathscr{U} = \{u \mid u_{\min} \le u \le u_{\max}\}, \tag{12}$$

$$\mathscr{X} = \{x \mid x_{\min} \le x \le x_{\max}\}. \tag{13}$$

The control algorithm includes swing leg control and ground force control as in work [8]. The first controller allows to follow the trajectory for the foot in the global frame and calculates joint torques as the sum of the feedback and feedforward terms.

In the second control, joint torques are generated using ground forces

$$\tau_i = J_i^\top R_i^\top f_i, \tag{14}$$

where J is the foot Jacobian, R is the rotation matrix, and vector f is computed using MPC approach.

A. Fully Connected Neural Network for MPC Weight Adaptation

FCNN, also known as a multilayer perceptron, is a type of artificial neural network where each neuron in one layer is connected to every neuron in the subsequent layer. An FCNN transforms input $\mathbf{x} \in \mathbb{R}^n$ to output $\mathbf{y} \in \mathbb{R}^m$ through sequential linear transformations and nonlinear activations. We used a simple 6-layer FCNN architecture, 24 input layer neurons, 4 hidden layers with 256 neurons, 13 neurons in output layer. All activation functions are ReLU, except for output layer, which has a sigmoid. For a simpler case, we used Q matrix only for adaptation. The input for FCNN is a Q weighting matrix used in MPC and a desired body state

$$\mathbf{x} = \begin{bmatrix}\text{vec}(Q)\\ x_{des}\end{bmatrix}. \tag{15}$$

The output layer is chosen as an error norm for each body state vector element and a norm of body state error norms, which represents the overall trajectory error

$$\mathbf{y} = \begin{bmatrix}||x_{\text{des}} - x_{\text{traj}}||^2\\ e\end{bmatrix}. \tag{16}$$

Other NN architectures may improve results, though this question was out of the scope of the presented research.

IV. SIMULATION RESULTS

In this section we investigate the dependency of a quadruped robot's behavior on the convex MPC weighting matrix Q. Parameters of the robot and MPC are given in Table I. By leveraging extensive simulations, a neural network and an optimization approach, the algorithm identifies optimal Q matrices for various desired velocities, enhancing trajectory tracking performance. Below, we provide a detailed explanation of each step in the proposed algorithm, outlining the methodology, computations, and optimization processes involved. The simulations are carried out using Raisim simulation environment (for instance, this software is applied by [16] for solving contact dynamics) and the FCNN is implemented in Python language. Here we consider the case in which the output layer in (16) is given by the values of Q matrix.

TABLE I
ROBOT AND MPC PARAMETERS

Parameter (units)	Value
Mass m (kg)	15.098
Center of mass COM (m)	(-0.01, 0, -0.021)
Inertia I (kgm^2)	diag$\{0.19, 0.49, 0.53\}$
MPC horizon length h (-)	15
MPC horizon time step dT (s)	0.0234

A. Simulation-based data collection

The first step involves generating a dataset that captures the relationship between the MPC weighting matrix Q, desired velocities, and the robot's trajectory tracking performance. We simulate 45,000 trajectories, each lasts 2 seconds. The diagonal elements of the MPC weighting matrix Q are randomly generated within the range of values 0.1-100. Desired velocities are constant during the trajectory and are randomly sampled within the XY plane, constrained to ±0.7 m/s in the x-direction and ±0.6 m/s in the y-direction. All random values were generated using uniform distribution. To ensure consistency, all trajectories start and stop at the beginning of a new gait phase, aligning with the robot's periodic motion. This setup guarantees that initial and final conditions are standardized, facilitating fair comparisons between simulations.

B. Dataset creation

For each simulated trajectory, we evaluate the tracking performance by computing error norms based on vector of body state, described in equation (6). For each element, we calculate the error norm, defined as the deviation between the actual and desired values over the trajectory. Additionally, we compute a composite norm of these error norms, which combines individual errors into a single scalar metric representing overall tracking performance. The resulting dataset consists of input-output pairs, where the inputs are the desired XY velocities and the diagonal elements of Q, and the outputs are the error norms for each state vector element and the composite norm.

C. Neural network training

To model the complex relationship between Q, desired velocities and trajectory errors, we train an FCNN. The dataset generated in the previous step is used for training, with the desired XY velocities and diagonal elements of Q as input features and the error norms (individual and composite) as output labels. FCNN is designed to predict the error norms for any given combination of desired velocity and matrix Q, effectively learning the underlying dependency without requiring further simulations.

D. Grid-based velocity discretization

To systematically explore optimal Q matrices across the velocity space, we discretize the XY velocity plane into a grid of nodes. Each node represents a specific combination of desired velocities within the defined limits (±0.7m/s in x, ±0.6m/s in y). For each node, we analyze the dataset to identify the trajectory with the best performance determined by a state error norm, then the trajectory's corresponding Q matrix and error norms are recorded. This step establishes a baseline of high-performing Q matrices for each velocity node, serving as a reference for further optimization. Generated surface with best trajectory cost function for each node of desired velocity set is presented in Fig. 1. It can be clearly seen that overall error tends to increase with higher desired velocities, which is likely due to model simplifications of dynamics.

E. Optimization of Q via backpropagation

Using the trained FCNN, we optimize the Q matrix for each velocity node, ensuring smooth transition. Starting from some constant Q matrix, we treat the error norms as the desired output of FCNN, and then iteratively adjust the input Q matrix for each node, using backpropagation through FCNN to minimize the difference between the predicted error norms and a target vector of error norms. As a target vector of error norms, we use error norms from the best trajectory for current velocity node. This optimization process leverages the gradient information provided by FCNN to refine Q, effectively tuning the MPC controller for each velocity node, ensuring smooth transition between nodes. The result is an optimized, effectively tuning the MPC controller for each velocity node.

F. Validation and comparison

For some specific velocity nodes, we simulate a trajectory using the optimized Q and compute the actual error norms, comparing them to the predictions from FCNN. This step verifies that FCNN's predictions are accurate and that the optimized Q matrices perform as expected in practice. Furthermore, we compare the tracking performance of the optimized Q matrices with a manually tuned Q matrix, which serves as a benchmark.

Tables II and IV show the weight coefficients, which were chosen empirically and generated by FCNN for different robot's velocities. Error norms for each scenario are given in Tables III and V. One can conclude that the control algorithm using weight matrix adaptation provides significant improvement in reducing the error rate. State trajectory error comparisons for empirically chosen and generated by neural network weights are presented in Figs. 2-5 for desired velocity $V_x = \dot{P}_x = 0.7$ m/s, $V_y = \dot{P}_y = 0$ m/s and in Figs. 6-9 for desired velocity $V_x = \dot{P}_x = 0$ m/s, $V_y = \dot{P}_y = 0.2$ m/s. As seen from figures, most state element's errors are decreased as well as composite error norm. For the first scenario the composite error norm is decreased by 14% compared to fixed weight matrix, and for the second velocity it is reduced by 45%.

V. CONCLUSIONS

This paper presents the model predictive control framework for quadrupedal robots, incorporating adaptive weight matrix tuning via the neural network. The proposed approach employs the single rigid-body dynamics model in the MPC

TABLE II

Q MATRIX VALUES FOR $V_x = 0.7$ M/S, $V_y = 0$ M/S

Method	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
Manual	30.00	30.00	400.00	70.00	70.00	50.00	100.00	100.00
FCNN	59.70	52.10	62.60	25.20	97.60	53.40	60.60	95.00

Method	Q_9	Q_{10}	Q_{11}	Q_{12}
Manual	100.00	7.00	7.00	1.00
FCNN	66.90	28.90	11.20	55.60

TABLE III

ERROR NORMS FOR $V_x = 0.7$ M/S, $V_y = 0$ M/S

Method	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
Manual	0.23	1.94	1.99	1.06	0.22	1.08	0.13	1.76	4.28	1.07
FCNN	0.14	3.86	0.51	1.13	0.11	0.97	0.12	1.08	2.15	0.93

Method	**C. Norm**
Manual	5.71
FCNN	4.91

formulation, where ground reaction forces at contact points govern the system's motion. The control strategy includes: i) swing-leg trajectory tracking for precise foot placement, ii) GRF optimization using whole-body impulse control for dynamic stability, and iii) neural-network-based adaptation of MPC weight matrices to enhance performance across varying locomotion speeds. The fully connected neural network (FCNN) dynamically adjusts the cost function weights in the convex MPC problem, enabling automatic adaptation to different gait conditions. Numerical simulations validate the method, demonstrating significant tracking error reduction compared to fixed-weight MPC across multiple robot speeds. The presented methodology considered MPC parameters, but also can be useful for adaptation of other parameters of the control algorithm.

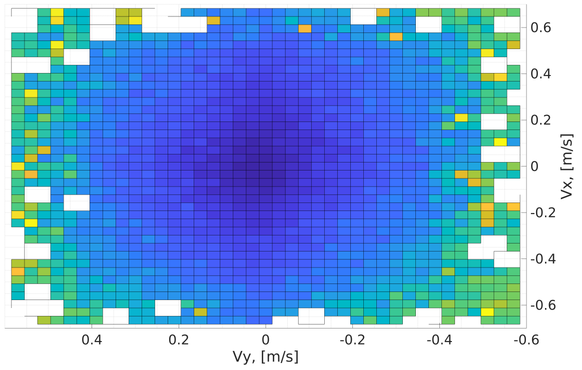

Fig. 1. Best trajectory cost function surface. Top view

REFERENCES

[1] M. H. Raibert, *Legged robots that balance*. MIT press, 1986.

[2] K. Afsari, S. Halder, M. Ensafi, S. DeVito, and J. Serdakowski, "Fundamentals and prospects of four-legged robot application in construction progress monitoring," vol. 2, pp. 274–283, 2021.

[3] H. Kolvenbach, D. Wisth, R. Buchanan, G. Valsecchi, R. Grandia, M. Fallon, and M. Hutter, "Towards autonomous inspection of concrete deterioration in sewers with legged robots," pp. 1314–1327, 2020.

[4] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," p. eaau5872, 2019.

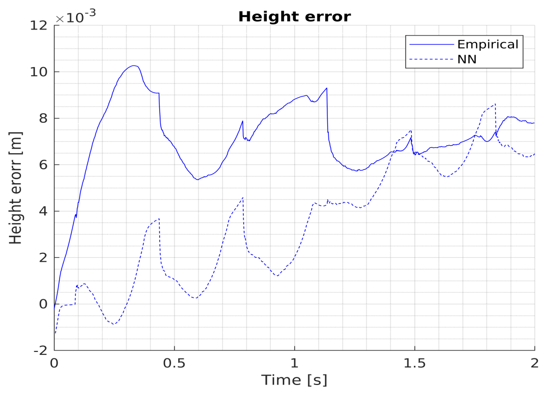

Fig. 2. Height error for $V_x = 0.7$ m/s, $V_y = 0$ m/s

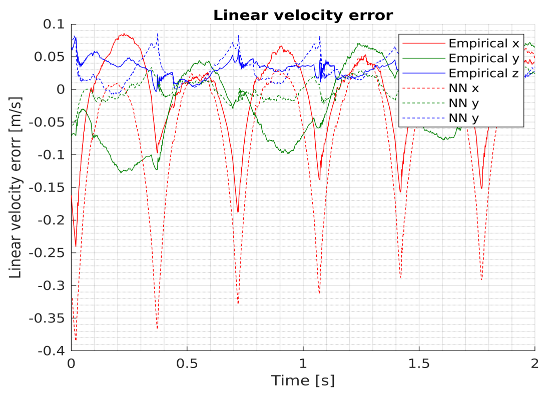

Fig. 3. Linear velocity error for $V_x = 0.7$ m/s, $V_y = 0$ m/s

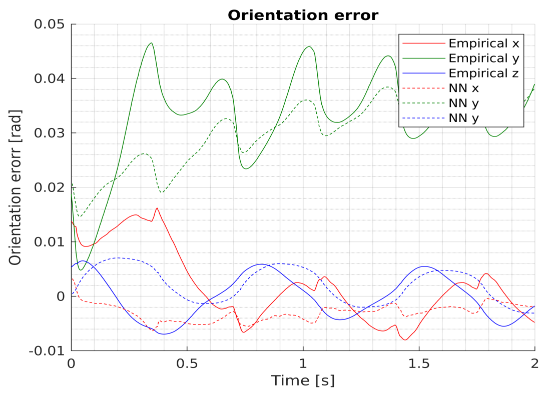

Fig. 4. Orientation error for $V_x = 0.7$ m/s, $V_y = 0$ m/s

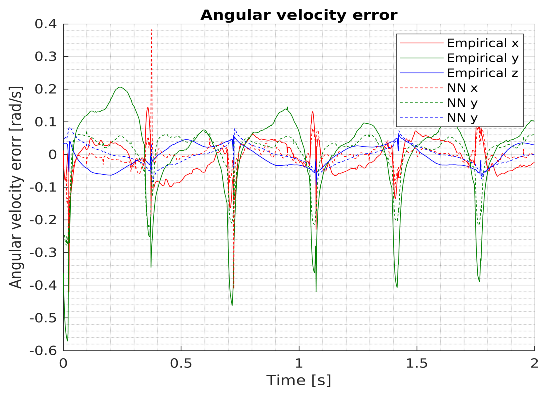

Fig. 5. Angular velocity error for $V_x = 0.7$ m/s, $V_y = 0$ m/s

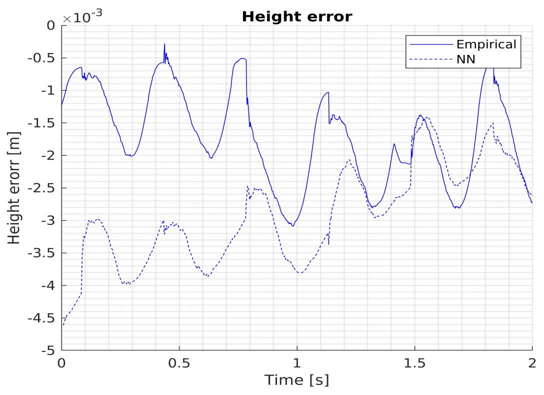

Fig. 6. Height error for $V_x = 0$ m/s, $V_y = 0.2$ m/s

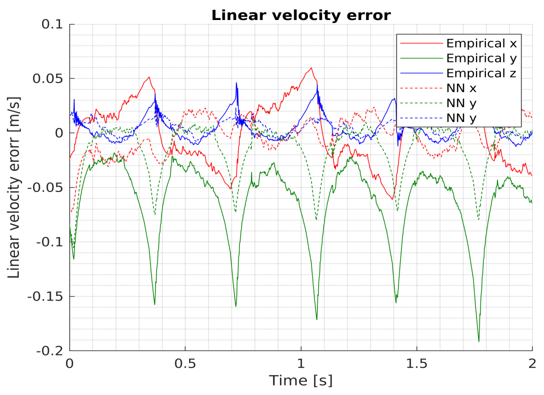

Fig. 7. Linear velocity error for $V_x = 0$ m/s, $V_y = 0.2$ m/s

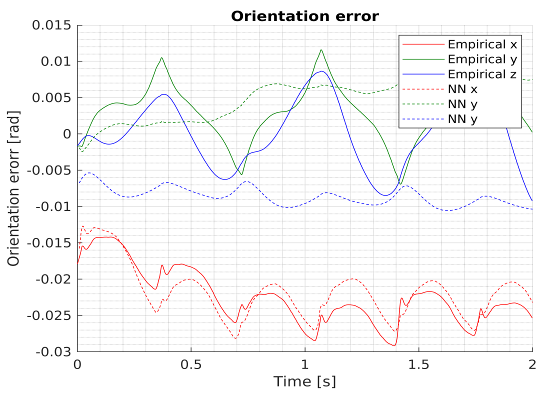

Fig. 8. Orientation error for $V_x = 0$ m/s, $V_y = 0.2$ m/s

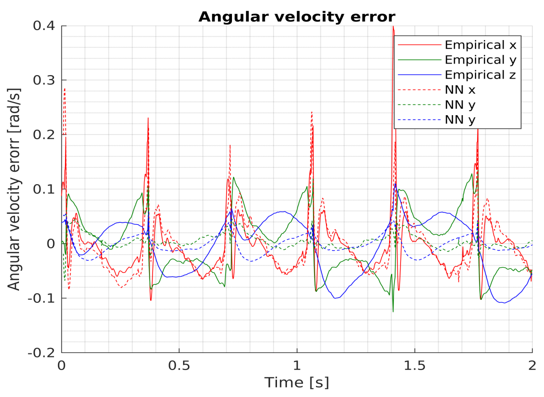

Fig. 9. Angular velocity error for $V_x = 0$ m/s, $V_y = 0.2$ m/s

TABLE IV
Q MATRIX VALUES FOR $V_x = 0$ M/S, $V_y = 0.2$ M/S

Method	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8
Manual	30.00	30.00	400.00	70.00	70.00	50.00	100.00	100.00
FCNN	40.00	37.90	65.50	16.60	70.90	51.90	103.90	27.70

Method	Q_9	Q_{10}	Q_{11}	Q_{12}
Manual	100.00	7.00	7.00	1.00
FCNN	51.70	6.60	49.70	50.70

TABLE V
ERROR NORMS FOR $v_x = 0$ M/S, $v_y = 0.2$ M/S

Method	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
Manual	0.06	1.02	2.27	0.43	0.72	0.17	0.16	1.79	1.80	1.67
FCNN	0.10	0.52	0.84	0.27	0.70	0.19	0.27	1.66	0.47	0.54

Method	**C. Norm**
Manual	4.03
FCNN	2.22

[5] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[6] N. Rudin, D. Hoeller, M. Bjelonic, and M. Hutter, "Advanced skills by learning locomotion and local navigation end-to-end," 2022.

[7] M. Chignoli and P. M. Wensing, "Variational-based optimal control of underactuated balancing for dynamic quadrupeds," pp. 49 785–49 797, 2020.

[8] J. Carlo, P. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," pp. 1–9, 10 2018.

[9] G. Bledt, P. M. Wensing, and S. Kim, "Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the mit cheetah," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 4102–4109.

[10] G. Bledt and S. Kim, "Extracting legged locomotion heuristics with regularized predictive control," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 406–412.

[11] M. Hutter, "Starleth & co.: Design and control of legged robots with compliant actuation," Ph.D. dissertation, ETH Zurich, 2013.

[12] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 38–44.

[13] J. Carlo, "Software and control design for the mit cheetah quadruped robots," Ph.D. dissertation, 01 2020.

[14] J. Nubert, "Learning-based approximate model predictive control with guarantees: Joining neural networks with recent robust mpc," 08 2019.

[15] T. Salzmann, E. Kaufmann, J. Arrizabalaga, M. Pavone, D. Scaramuzza, and M. Ryll, "Real-time neural MPC: Deep learning model predictive control for quadrotors and agile robotic platforms," pp. 2397–2404, 2023.

[16] J. Hwangbo, J. Lee, and M. Hutter, "Per-contact iteration method for solving contact dynamics," pp. 895–902, 2018. [Online]. Available: www.raisim.com