Università degli Studi di Catania

Dottorato di Ricerca in Ingegneria dei Sistemi, Energetica, Informatica e delle Telecomunicazioni

Neuro-inspired legged robots in unstructured terrains: from locomotion to data-driven exploration

Relatore:
Prof. Paolo Arena
Correlatore:
Prof. Luca Patanè

Candidato:
Salvatore Taffara

Anno Accademico 2021–2022

Ai miei genitori
per la loro costante dedizione nel rendermi felice.

Abstract

This PhD thesis aims to analyze research topics related to neural-inspired solutions for robotic locomotion in order to provide new tools and answers to a series of still open questions.

The use of quadrupedal robots, thanks to their unique abilities, represents one of the most debated research topics in recent years. Neural methods are applied from the very low level of the direct control of the single joint of a robot leg, up to the highest level to model the whole robot behaviour and success while accomplishing a given task.

In particular, analysis will be carried out regarding new central pattern generator methods based on the reaction-diffusion mechanism, FitzHugh–Nagumo's neuron and sensory feedback. Neural intelligence techniques will be used for the ground reaction forces estimation, the motor faults compensation and the ground slope estimation with the aim to obtain an efficient robot state estimation method giving the possibility to make the robotic structure more robust and performing. Considering advanced MPC-based stability controls, the usage of neuro-inspired control gait mechanisms, with respect to simpler linear approaches, will make significant differences in terms of the capabilities of the robot to follow a reference trajectory. An overview is then reported regarding the mobile robot energetic consumption. Finally, the topic of robotic navigation in unstructured terrains will be considered. Given an unstructured complex terrain and a group of robots (wheeled, quadrupeds, hybrid, and so on), each of these unique in its characteristics and traversing capabilities, the goal is to find which paths, among the infinite ones within the terrain, they can cope according to their current mechanical abilities, skills, and knowledge of the environment.

Contents

Chapter 1

Introduction

The world of bio-inspired robots, especially in recent decades, has become increasingly popular due to the fact that researchers and scientists have been able to verify the extraordinary world that regulates the actions and activities of insects and animals. Insects show a lot of amazing innate behaviors, for example, they cooperate when they have to complete a given task or they are able to find an optimal path in a complex environment from a starting point to an end point using chemical messages entrusted to substances called pheromones. Besides, the limited number of contact points with the ground gives them the possibility to reduce friction whereas the use of efficent control strategies allows to actuate a balancing control even in complex unstructured terrains by regulating the angles between their legs. All these aspects have been applied to insect-inspired and quadruped robots in the last years by researchers around the world.

Legged locomotion provides important advantages for the exploration of uneven terrain, in particular in presence of obstacles on their path. The first attempts to reproduce gait on legs in robots were aimed at the use of hexapods [1, 2] and octopus robots [3], to assess locomotion stability, guaranteeing always at least three contact points with the ground. In [4] legged robots are used for the consolidation of buildings and structures in general, while in [5] the problem of the morphological changes in terrain was faced using a robot with wheels and front legs able to deposit sensors in the ground. The steps forward in the electronic and sensor fields have given the opportunity to have hardware components, like sensors and actuators, with low cost that have made mobile robots smarter with respect to the past. For all these reasons, quadrupedal locomotion is starting to be employed, guaranteeing a high level of robustness to disturbances and, at the same time, permitting a relevant improvement in the robot's dexterity.

The goal of this thesis is to face the topic of the legged control mechanism, from a low-level to a high-level point of view. For this reason, the works reported in the following chapters can be grouped into different research lines which, converging together, address the issue of bio-inspired robot locomotion in unstructured terrains.

1.1 From a low to a high level analysis

Starting from a low-level analysis of the problem, examples of locomotion control using Central Pattern Generator (CPG) approaches for the robotic locomotion, are reported. It is well known that CPG is a paradigm for locomotion control which aims in attaining a specific locomotion gait without, in principle, considering sensor feedback as strictly needed. The main issue is to reach a stable phase displacement among the legs which guarantees an arbitrary steady state synchronization, given some hyphotheses. The partial contraction theory will be used as a locomotion stability guarantee. Two different CPG approaches will be formalized and applied to a hybrid legged and wheeled robot, in particular, a reaction-diffusion based one and a FitzHugh–Nagumo's neuron based one, then, a focus on the role of the environmental feedback is provided.

Starting from a reliable CPG locomotion controller, neural approaches to estimate Ground Reaction Forces (GRFs) for proprioception with the usage of SNNs (Spiking Neural Networks) are proposed. Proprioception is the ability of a robot to perceive and recognize its body position in space, without the support of sensors; it is considered a sixth sense as it is regulated by a specific part of the brain. SNN are demonstrated to be more efficient than non-spiking networks. CPG and active proprioception constitute an added value to adaptive locomotion. In this way, it is possible to predict robot intrinsic quantities, such as GRFs, starting from other quantities taken from motors, like Torques (TRQs), even if one or more motors are damaged. At the end of this part, it will be possible to design an efficient robot state estimation method based on Reservoir Computing (RC) theory.

From the locomotion control aspect, the design of a linear and a nonlinear MPC controller for a quadruped robot endowed with FitzHugh-Nagumo CPG is proposed. In particular, comparisons will be carried out between the results obtained using different control mechanisms such as Proportional Integrative Derivative (PID), Model Predictive Control (MPC), and Neural Network Model Predictive Control (NNMPC). It will be reported demonstrating, in particular, that NNMPC represents the preferable control typology in case of slippery terrains.

It is well known that legged locomotion involves, in principle, a large energy consumption over the wheeled approach. For this reason, an energy analysis evaluation, expressed using an univocal index, will be presented evaluating the quadruped locomotion efficiency in unstructured environment in comparison with other simulated structures. The index is the Cost of Transport (CoT) which will be used to estimate how expensive one path is compared to another and how, for the same paths, two robots give different results due to their different structures: this is fundamental from an engineering resource minimization point of view.

From a high-level side, topics related to risk-based path planning in unstructured environments will be presented. In particular, a discussion will be carried out about a data-driven approach for learning risk-mediated traversability maps in the case of multiple robot architectures and multiple paths characterised by different levels of complexity and associated risk. Here, the method for obtaining a neural model representing the unique characteristics of a mobile robot is presented considering also an acceptable risk threshold associated with the path. In this way, at the end of this chapter, an optimal path estimator model and risk threshold based is provided.

1.2 Novelties

The main novelties presented in this thesis can be schematized as follows:

CPG methods for the quadruped locomotion

- Application to the quadruped locomotion of structures such as reaction-diffusion system, FitzHugh–Nagumo's neuron, Matsuoka's neuron and SO(2)-networks.
- Starting from the mathematical formalizations of the structures presented above, different CPG structures are obtained and then applied to the robotic field.

Liquid State Machine for the Ground Reaction Forces prediction

- Creation of an efficient robot state estimation method based on reservoir computing: from the local proprioceptive information acquired at the level of the leg joints (torques) of a simulated quadruped robot it is possible to obtain the GRFs recorded at the foot level.
- Comparative analysis between the Echo State Network (ESN) and Liquid State Machine (LSM) approaches.

Evaluation of robot energy consumption

- Implementation of a simple robotic control strategy based on the FitzHugh–Nagumo Neuron (FHN) used as a basic block for the implementation of a CPG.
- The possibility to apply a nullcline-based control strategy to adopt the neuron oscillation proprieties also facilitating the synchronization between the different legs.
- Focus on an adaptative control to minimize the energetic cost.

MPC-based control strategies for the gait of a neuro-inspired quadruped robot

- Implementation of a neural adaptative structure based on the proprioceptive information and the exteroceptive signals acquired through ground contact sensor.
- Locomotion control realised using a central pattern generator implemented through oscillators synchronized through environmental feedback.
- Results compared with those obtained using a linear MPC and a PID base controller.

Robot-oriented neural models for path planning

- Design of a simple procedure for the generation of traversability maps needed for the definition of the optimal path between two or more target positions. To avoid the wearing out of the robots, they are simulated in an accurate dynamic environment, and their traversing capabilities are learned through a data-driven approach.
- The approach produces a robot-specific traversability map, directly derived from the robot exploration.

- A multi-robot approach is used, where different robots can fulfil different missions according to their kinematic and dynamic capabilities that are not analytically defined but are inferred from the simulation data.
- Different traversability map estimation methods are developed considering not only the minimisation of the travelled distance but also the risk related to the specific path.
- The low computational complexity applied to real map terrains is a key aspect in view of the implementation on embedded architectures for edge computing.

1.3 Software environments

The software tools employed in the following chapters are MATLAB, CoppeliaSim, Nest simulator, and ROS simulator.

CoppeliaSim [6] is a dynamic simulation environment that also hosts different robotic structures, robotic elementary parts, and environments in which robots can interact. The CoppeliaSim framework provides an accurate and realistic dynamic simulation environment where performance can be evaluated before implementation on the actual robot prototype. This simulation approach becomes essential when a large amount of data has to be acquired, i.e., when data-based learning is involved.

Nest simulator [7] is a spiking neural network simulator used for different purposes involving the neural science area. It can be adopted for the information processing in the visual or auditory cortex of mammals, for the implementation of models of network activity dynamics, e.g., laminar cortical networks or balanced random networks, models of learning and plasticity and so on.

The ROS (Robot Operating System) [8] serves as the interface for the robot. It provides libraries and tools to help software developers to create robot applications. ROS simulator, together with the Nest simulator, will be used to build the LSM architecture.

1.4 Chapter outline

The following thesis is organized as follows. The topics related to the low-level analysis are reported in:

- **Chapter 2**: the mathematical and formal description of the statistical indexes used in this thesis is here reported.
- **Chapter 3**: new CPG approaches based on reaction-diffusion system and FitzHug-Nagumo's neuron applied to legged robots sensory feedback.
- **Chapter 4**: GRFs estimation using LSMs carrying out comparisons between different methodologies.
- **Chapter 5**: evaluation of energetic efficiency in an unstructured environment and test on the real robot.

The remaining topics, related to the high-level analysis, are faced in:

- **Chapter 6**: design of a linear and a nonlinear MLP controller for a quadruped robot.
- **Chapter 7**: neural-based methods for the derivation of the traversability maps of unstructured environments.

In **Chapter 8**, the conclusions and future developments are reported.

Chapter 2

Performance analysis indexes

In this section, the formal description of the statistical indexes used in the following chapters is reported. In particular, the indexes are:

- Accuracy, sensitivity, and specificity.
- Maximum Absolute Error.
- Normalized Root Mean Square Error.
- Mean Squared Error.
- Goodness Of Fit based on Normalized Root Mean Square Error.
- Akaike Information Criterion.
- Stability and harmony.

2.1 Accuracy, sensitivity, and specificity

The accuracy index indicates how close a given set of measurements (observations) are to their reference values, i.e. accuracy is the proximity of measurement results to the reference value [9]. In Fig.2.1 the graphical representation of the accuracy index related to a reference index is shown.

The sensitivity and specificity are two statistical indexes describing the goodness of a test in terms of results when a condition to be fulfilled is imposed. Individuals for which the condition is satisfied are considered "positive" and those for which it is not are considered "negative". Sensitivity (true positive rate) refers to the probability of a positive test, conditioned on truly being positive. Specificity (true negative rate) refers to the probability of a negative test, conditioned on truly being negative [10]. In Fig.2.2 the formulas to obtain the sensitivity and specificity indexes are reported. In particular:

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.1a}$$

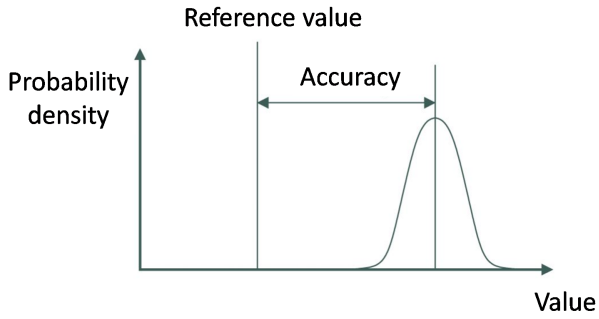


Figure 2.1: The accuracy related to a reference value considering a Gaussian measurements distribution. The probability density indicated is a function that provides the likelihood that the value of a random variable will fall between a certain range of values.

$$Specificity = \frac{TN}{TN + FP} \tag{2.1b}$$

where:

- **TP (True Positive)**: elements that are truly positive and that are predicted as such.
- **FP (False Positive)**: elements that are truly negative but that are predicted as positive.
- **FN (False Negative)**: elements that are truly positive, but that are predicted as negative.
- **TN (True Negative)**: elements that are truly negative and that are predicted as such.

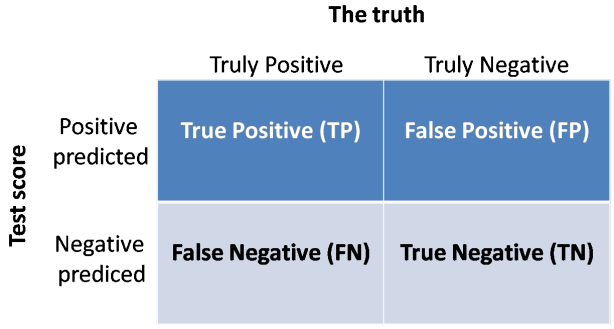


Figure 2.2: Diagram demonstrating the basis for deriving the elements in the sensitivity and specificity indexes.

2.2 MaxAE

The Maximum Absolute Error (MaxAE), given a series of measurements, represents the maximum difference between the measured or inferred value of a quantity y_t and its actual value y, it is given by:

$$MaxAE = max(\delta y) \tag{2.2}$$

with $\delta x = y_t - y$.

2.3 Mean Squared Error

The Mean Squared Error (MSE) index is defined as follows:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_{t_i})^2}{n} \tag{2.3}$$

where y_i is the prediction, y_{t_i} is the reference value and n is the number of considered patterns. It measures the average squared difference between the estimated values and the actual value and it corresponds to the expected value of the squared error loss.

2.4 Normalized Root Mean Square Error

The Normalized Root Mean Square Error (NRMSE) is defined as follows:

$$E(y, y_t) = \sqrt{\frac{\langle \|y_t(n) - y(n)\|^2 \rangle}{\langle \|y_t(n) - \hat{y}_t(n)\rangle\|^2 \rangle}} \tag{2.4}$$

where y is the prediction, y_t is the reference value, $\hat{y}_t$ is a theoretical value, n is the number of considered patterns and $\| \cdot \|$ stands for the Euclidean norm. It represents the normalized version of the Root Mean Square Error (RMSE) that is a measure of the differences between values predicted by an estimator and the values observed.

2.5 Goodness Of Fit based on Normalized Root Mean Square Error

The Goodness Of Fit (GoF) index is based on the NRMSE index and represents the error norm between the model output y in the testing data set and the corresponding reference signal y_t:

$$Fit = \Big(1 - \frac{||y_t(n) - y(n)||}{||y_t(n) - \hat{y}_t(n)||} \cdot 100\Big) \tag{2.5}$$

where $\hat{y}_t$ is a theoretical value [11] and n is the number of considered patterns

2.6 Akaike Information Criterion

The Akaike Information Criterion (AIC) estimates the model prediction quality, given a collection of data-driven models, based on the obtained prediction accuracy and model complexity [12]. According to Akaike's theory, the most accurate model should have the smallest AIC index, defined as follows:

$$AIC = N \cdot \log\left(\det\left(\frac{1}{N}\sum_{1}^{N}\epsilon\big(t,\hat{\theta}_N\big)\big(\epsilon\big(t,\hat{\theta}_N\big)\big)^T\right)\right) + 2n_P + N\cdot\big(n_y\cdot\big(\log(2\pi)+1\big)\big) \tag{2.6}$$

where N is the number of patterns in the estimation dataset, $\epsilon(t)$ is a $n_y - by - 1$ vector of prediction errors, θ_N represents the estimated parameters, n_p is the number of parameters to be estimated (model complexity), n_y is the number of model outputs.

2.7 Stability and harmony

Stability and harmony are two indexes used to evaluate the property of a moving object in a plane [13, 14], in particular:

- **Stability** accounts for the robot deceleration and rotational oscillation. It is defined as follows:

$$Acc_{min} = |\min(Acc)| \tag{2.7a}$$

$$Ang_{PP} = \max(Ang) - \min(Ang) \tag{2.7b}$$

$$a_{min} = \max(Acc_{min}) \tag{2.7c}$$

$$PP_{min} = \max(Ang_{PP}) \tag{2.7d}$$

$$A_{min_{final}} = \max(A_{min}) \tag{2.7e}$$

$$PP_{final} = \max(PP) \tag{2.7f}$$

$$Stability = \frac{e^{-\lambda A_{min_{final}}} + e^{-\lambda PP_{max_{final}}}}{2} \tag{2.7g}$$

where Acc indicates the accelerations of the robot body and Ang the angular velocities in all three axes. The values of Acc minimum peak (Acc_{min}) and Ang peak to peak (Ang_{PP}) characterize the locomotion stability. Stability is evaluated during the time window of the experiment. This is divided into several cycles or periods ($N_{periods}$). We define $A_{min} = \{a_{min,1};\ ....;\ a_{min,N_{periods}}\}$, $PP = \{PP_{min,1}\ ;....;\ PP_{min,N_{periods}}.\}$ The decay constant λ is fixed to $\lambda = 0.1$.

- **Harmony** indicates the adaptation of the robot body position to the change in the terrain level. To evaluate the Harmony index, the following relations are calculated:

$$RI_{a_{min}} = \frac{\min(Acc_{min})}{\max(Acc_{min})} \tag{2.8a}$$

$$RI_{PP_{min}} = \frac{\min(Ang_{PP})}{\max(Ang_{PP})} \tag{2.8b}$$

$$RI_{A_{final}} = \min(RI_A) \tag{2.8c}$$

$$RI_{PP_{final}} = \min(RI_{PP}) \tag{2.8d}$$

$$Harmony = \frac{RI_{A_{final}} + RI_{PP_{final}}}{2} \tag{2.8e}$$

where $RI_A = \{RI_{a_{min,1}}; RI_{a_{min,2}}; ....; RI_{a_{min,N_{periods}}}.\}$ and $RI_{PP} = \{RI_{PP_{min,1}}; RI_{PP_{min,2}}; ....; RI_{PP_{min,N_{periods}}}.\}$.

Chapter 3

CPG methods for the quadruped locomotion

In this chapter, innovative CPG approaches for the robotic locomotion will be reported. In particular, the CPGs are based on the following structures:

- Reaction-diffusion system
- FitzHugh–Nagumo's neuron
- Matsuoka's neuron and environmental feedback
- SO(2)-networks as neural oscillators

3.1 Reaction-diffusion system

The locomotion control of a quadruped or hybrid robot is a challenging task. In the animal kingdom, from mollusks to men, it is a clear evidence that locomotion flow, in steady state conditions, involve stereotyped limbs dynamics which take place, in principle, in the absence of sensory feedback. Legs are controlled by centralised neural controllers which send to the locomotion actuators robust phase synchnised oscillatory signals. This is mostly evident in escape reactions: whichever are the ground conditions running is implemented mostly invariantly. A fast and reactive response suggests the application of a centralized control system. In particular, in [1] the authors employ Partial Contraction Theory to demonstrate asymptotic stability in Reaction Diffusion based oscillatory networks, and in [15] the high level locomotion control, involving steering, was argued to preserve stability by weighting the oscillator states. Here this result if formally demonstrated.

A reaction-diffusion system-based CPG architecture was formalized in [15] and in [5] and it is applied to a hybrid legged and wheeled robot. Cellular Neural Networks (CNNs) are arrays of simple, identical, locally interconnected nonlinear dynamic circuits called cells to build impressive analog signal processing systems [16]. When applied to locomotion control, the main focus is to look for steady-state solutions able to guarantee

an asymptotically stable equilibrium point of the phase solutions among the cells locally behaving as nonlinear oscillators. The dynamics of the single node, for this particular application, are described by the following two differential equations:

$$\begin{cases} \dot{x}_{1,i} = f_1(x_{1,i}, x_{2,i}) = -x_{1,i} + (1+\mu)y_{1,i} - sy_{2,i} \\ \dot{x}_{2,i} = f_2(x_{1,i}, x_{2,i}) = -x_{2,i} + (1+\mu)y_{2,i} - sy_{1,i} \\ y_{k,i} = \tanh(x_{k,i}) \;\; here \;\; k = 1, 2 \end{cases} \tag{3.1}$$

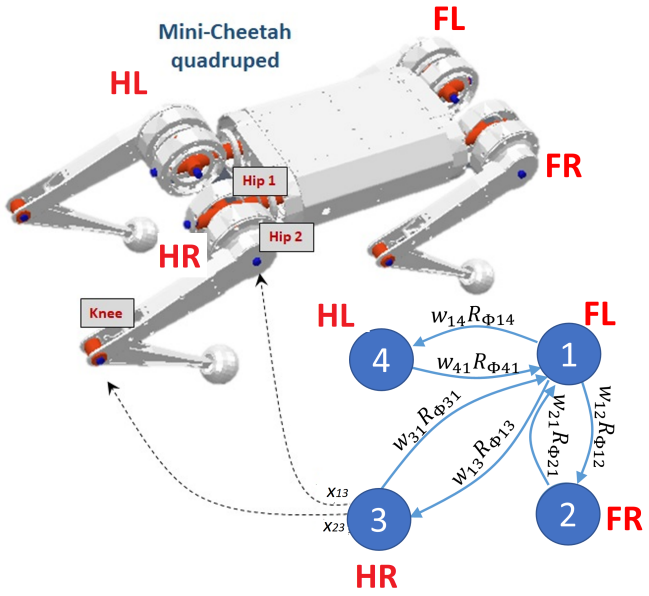


Figure 3.1: The neural structure adopted for locomotion control. Each neuron is devoted to control the position of two leg joints (Hip 2 and Knee) through the two-state variables $x_{1,i}$ and $x_{2,i}$ defined in Eq. 3.1. As example, the two-state variables $x_{1,3}$ and $x_{2,3}$ controlling the Hip 2 and Knee related to the 3^{rd} leg are reported.

where parameters $\mu = 0.7$ and $s = 1$ were selected in order to obtain a stable limit cycle [15].

Connecting n identical cells, the whole system dynamics can be thus represented in the following general form:

$$\dot{\mathbf{x}}_\mathbf{i} = f(\mathbf{x}_\mathbf{i}) + k_L \sum_{i \neq j \;\; j \in N_r} [R_{ij}(\mathbf{w_j x_j}) - \mathbf{x_i}] \tag{3.2}$$

where:

- $f(\mathbf{x_i}) = f(x_{1,i}, x_{2,i})$ represents the dynamics of the $i-th$ uncoupled oscillator.

- k_L is the feedback gain, whereas the summation operator represents the contribution of the immediately neighboring cells around the cell i, within a neighbour N_r.
- The last part of the equation represents the feedback error between the state variables of the $i-th$ cell and the corresponding state variables of cell j, eventually after a scaling, through the weight vector $\mathbf{w_j}$, and a phase-shift with respect to cell i via the rotation matrix R_{ij}.
- The value $\mathbf{w_j}$ represents a scaling weight vector on the state variables of certain cells, that will enable to easily implement a gain-based steering control [17].

Under these conditions, the two oscillators attain reciprocal equilibrium by minimising the error term, which implies phase-shift weighted synchronization.

The dynamics of the whole locomotion controller, shown in Eq. 3.2, can be expressed in a more compact form by the following reaction-diffusion system, outlining the Laplacian operator:

$$\dot{x} = f(x) - k_L \cdot L \cdot (w \odot x) \tag{3.3}$$

where:

- $x = [x_{1,1}, x_{2,1}, ..., x_{1,n_n}, x_{2,n_n}]^T \in R^{n_t}$.
- $f(x) = [f(x_{1,1}, x_{2,1}), ..., f(x_{1,n_n}, x_{2,n_n})]^T$.
- $w \in R^{n_t}$ is a weight vector acting element-wise on the state variables.
- $L \in R^{n_t \times n_t}$ is the laplacian matrix built-up of blocks $R_{i,j}$ (i.e. rotational matrix) whose values are based on the topology of the network and the specific gait to be generated.
- The gain k_L used to weight the laplacian coupling among neurons, introduced in Eq. 3.2, can be selected in order to satisfy the sufficient condition imposed by exploiting Partial Contraction theory [18].

This will guarantee the convergence towards a desired synchronization pattern that represents a stable locomotion gait for the quadruped structure [1, 18].

In order to build the Laplacian connection matrix L, considering any two neurons i and j $(i, j = \{FL, FR, HL, HR\})$, the L sub-matrix $L_{i,j}$ between them assumes one of the following forms:

$$\begin{cases} L_{i,j} = -R(\theta_{i,j}), \textit{ if cells i and j are connected with a phase shift } \theta_{i,j}. \\ L_{j,i} = -R(-\theta_{i,j}), \textit{ for undirected balanced networks.} \\ L_{i,j} = L_{j,i} = 0, \textit{ if no connection does exist between nodes i and j.} \\ L_{i,i} = d_i I_{n_c}, \forall i = 1..n_n, d_i\textit{: unweighted degree of node } i^{th}. \end{cases} \tag{3.4}$$

This theory formalises the problem of phase shift synchronization through the introduction of a flow-invariant subspace $\mathcal{M}$, i.e. a subspace where the system trajectories are

to be trapped. $\mathcal{M}$ encodes the phase shifts among the state variables. More in details, for undirected diffusive tree graphs with $w = 1$ in Eq.3.3, as shown in [1, 18], a unique gain K_L can be found to guarantee the global exponential convergence to any imposed phase shift among the neurons, i.e. any imposed locomotion gait.
However, for general directed diffusive weighted graphs, the laplacian matrix is generally non-symmetric and unbalanced, so it is not possible to apply directly the theoretical results already proven.

For the structure considered in this chapter, as demonstrated below, it is possible to find a suitable transformation to reformulate our graph in terms of an underlying undirected diffusive tree graph. So, it is still possible to apply the partial contraction result to demonstrate the convergence to a flow invariant subspace: this result was only argued in the simulation results in [17]. For this purpose, let us prove the following Lemma.

Lemma 1 The non-symmetric connection matrix of the directed weighted graph in Fig.3.1 is similar to a symmetric Laplacian matrix. Therefore, the directed weighted graph in Fig.3.1, can be reformulated in terms of an undirected diffusive tree graph. To demonstrate such a lemma, let us first explicit Eq. 3.3, for the graph in Fig.3.1. It holds:

$$\begin{cases} \dot{x}_1 = f(x_1) + K_L \sum_{i=1}^{4} [R_{\Phi i1}(w_{i1}x_i) - x_1] \\ \dot{x}_2 = f(x_2) + K_L [R_{\Phi 12}(w_{12}x_1) - x_2] \\ \dot{x}_3 = f(x_3) + K_L [R_{\Phi 13}(w_{13}x_1) - x_3] \\ \dot{x}_4 = f(x_4) + K_L [R_{\Phi 14}(w_{14}x_1) - x_4] \end{cases} \tag{3.5}$$

Let us now recall that, for every rotation matrix R_ϕ, any weight $w_{ij} \in R$ and any vector $x \in R^2$, it holds: $R_\phi(wx) = wR_\phi(x)$. The scaling gain of the state variables of cell j with respect to cell i can be seen as a weight on the $i \to j$ link. Reformulating the equations above, outlining the laplacian matrix (Eq.3.3), it holds:

$$\hat{L} = \begin{bmatrix} 3I & -w_{21}R_{\phi 21} & -w_{31}R_{\Phi 31} & -w_{41}R_{\Phi 41} \\ -w_{12}R_{\phi 12} & I & 0 & 0 \\ -w_{13}R_{\phi 13} & 0 & I & 0 \\ -w_{14}R_{\phi 14} & 0 & 0 & I \end{bmatrix}$$

Since, in general $w_{ij} \neq w_{ji}$ $\hat{L}$ is neither symmetric nor balanced[1].

Let us now define the following balancing non-singular matrix:

$$T = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & \sqrt{\frac{w_{21}}{w_{12}}}I & 0 & 0 \\ 0 & 0 & \sqrt{\frac{w_{31}}{w_{13}}}I & 0 \\ 0 & 0 & 0 & \sqrt{\frac{w_{41}}{w_{14}}}I \end{bmatrix}$$

It is easy to verify that $T\hat{L}T^{-1} = L$ with:

[1]A network (and so its Laplacian matrix $L = \{l_{ij}\}$) is balanced if:$\sum_{i \neq j} l_{ij} = \sum_{i \neq j} l_{ji}$

$$L = \begin{bmatrix} 3I & R_{\phi 21} & -R_{\Phi 31} & R_{\Phi 41} \\ -R_{\phi 12} & I & 0 & 0 \\ -R_{\phi 13} & 0 & I & 0 \\ -R_{\phi 14} & 0 & 0 & I \end{bmatrix} \tag{3.6}$$

Supposed that $R_{\phi ij} = -R_{-\phi ji}$, [2], L is the symmetric, balanced Laplacian matrix of an undirected diffusive tree graph underlying the directed graph of Fig.3.1.

It is possible to demonstrate the following theorem:

Theorem 1 The directed weighted graph of Fig.3.1, represented by Eq. 3.3, possesses solutions exponentially converging to any imposed flow-invariant subspace $\mathcal{M}$, (i.e., to any phase shift among the oscillators) if there exists one gain $K_{min} : \forall k \geq K_{min}$ it holds:

$$k \cdot \lambda_1 \ > \sup_{x_i} \lambda_{\max} \left(\frac{\partial f}{\partial x}(x_i, t) \right) \tag{3.7}$$

λ_1 is defined as the algebraic connectivity of the graph [19], i.e. the smallest non-vanishing eigenvalue of the Laplacian matrix:

$$\lambda_1 = \lambda_{min}(L) = \lambda_{min}(\mathcal{V}^T \cdot L \cdot \mathcal{V})$$

being $\mathcal{V}$ the orthonormal complement associated with the flow-invariant subspace $\mathcal{M}$; the second member in Eq.3.7 represents the maximum eigenvalue of the Jacobian matrix of the generic uncoupled cell.

The proof is based on the results from Lemma 1 and of Theorem 1 in [1]. In fact, for the weighted directed tree graph of Fig.3.1, Lemma 1 states that there exists an underlying undirected weight-balanced diffusive tree graph characterised by the balanced symmetric, positive semi-definite laplacian matrix defined in Eq.3.6. According to Theorem 1 in [1], for any imposed flow-invariant subspace (and therefore for the corresponding associated Laplacian matrix), representing any given phase shift among the oscillators arranged into an undirected diffusive tree graph, there exists a minimum gain k_L for which Eq.3.7 holds.

Theorem 1 in [1], is thus extended to the weighted directed tree graph case herewith presented, where it is now possible to define any phase among the oscillators and any scaling among them, being assured the existence of an asymptotically stable solution to the imposed phase equation. It has to be noticed that, although not general, the case of this weighted directed tree graph underlines a lot of network structures, where an arbitrary phase shift synchronization is desired among oscillators of different amplitudes.

The steering control of the robot, once selected the proper stabilising gain k, can be performed by tuning the weights w_{ij} in Eq. 3.3; this result will be applied in Section 6.3.3.

Considering Eq.3.2, in [5] this latter is applied to a hybrid legged-wheeled robot, and here, the system equation is modified in order to create a steering. This is obtained by connecting the oscillators using direct links and imposing a phase relation through the rotational matrix. For instance, we want that the left and right legs are in anti-phase during locomotion, an $R(\phi)$ with $\phi = 180^o$ can be applied in Eq.3.2. In order to

[2] This simply means that if cell i is lag phase shifted of a given angle Φ on cell j, this latter cell is lead shifted of the same angle Φ with respect to cell i.

implement a steering, the implementation of a time-varying rotational matrix $R(\phi(t))$ is realized. The steering of the robot is realized using different timing factors t associated with the rotational matrix. Fig.3.2 shows the robot trajectory performed with different rotational matrices that generate distinct curvature radii. The inner side leg stepping amplitude is reduced and, at the same time, the inner side wheel speed is decreased acting on the time-dependent rotational matrix $R((t))$.

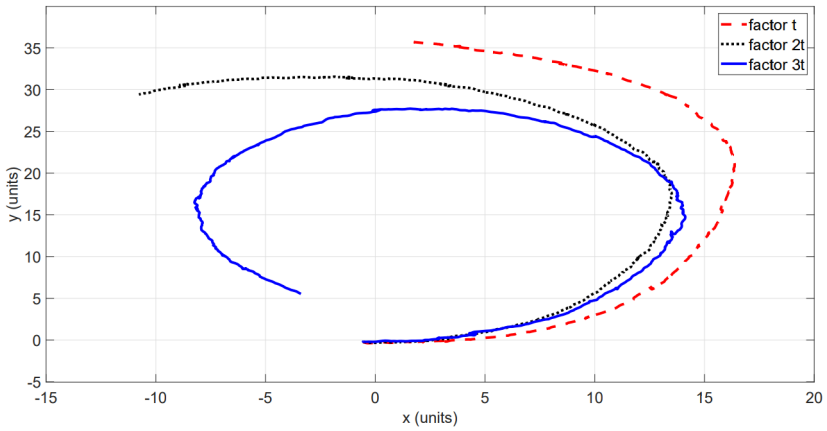


Figure 3.2: Steering of the robot based on different timing factors t associated with the rotational matrix.

3.2 FitzHugh–Nagumo's neuron

In[20], a CPG is applied to the control of a simulated Mini Cheetah robot ([21, 22]) using a network based on the FitzHugh–Nagumo's neuron [23]. The locomotion patterns are generated using a series of neurons connected, shown in Fig.3.3, controlling eight joints related to the hip and knee of each leg (the coxa joint is not exploited in this work where only sagittal movements of the legs are considered). The control system is developed using four FHNs able to produce oscillating, rhythmic patterns in the absence of sensory feedback, connected together to implement a CPG able to synchronize the oscillation phases to obtain the desired locomotion gait [24]. As we can see, the network graph is identical to that in Fig.3.1, so that we can, also in this case, apply the same approach for the robot steering.

The FHN's neuron mathematical model consists of two coupled, nonlinear ordinary differential equations, modelling the fast evolution of the neuronal membrane voltage and the slower recovery action of the sodium channel and potassium channel deactivation. The cubic non-linearity present in the original model is approximated using a Piece-Wise Linear function (PWL) obtaining what is called McKean's caricature of the FHN [25]. The equation describing the behaviour of the FHN's neuron is reported in the following:

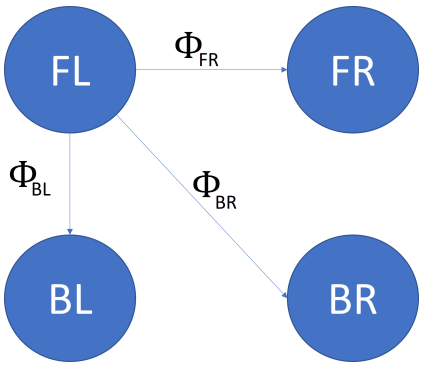


Figure 3.3: CPG structure. The legs are BL: back left, BR: back right, FL: front left, and FR: front right. The synchronization phase between neurons is related to the adopted locomotion gait.

$$\begin{cases} \dot{x}_1 = f_1(x_1, x_2) = (\frac{1}{\varepsilon}(\gamma(x_1) - x_2)) \\ \dot{x}_2 = f_2(x_1, x_2) = (x_1 + a - bx_2) \\ \gamma(x) = a_0 + a_1 x + \sum_{j=1}^{N} b_j |x - E_j| \end{cases} \tag{3.8}$$

where:

- a, b, $\varepsilon \in R$.
- $N = 2$.
- $a_0 = \gamma(0) - b_1|E_1 - b_2|E_2$.
- $a_1 = \frac{1}{2}(m_0 + m_2)$.
- $b_1 = \frac{1}{2}(m_1 - m_0)$.
- $b_2 = \frac{1}{2}(m_2 - m_1)$.
- $E_j \in R$.
- m_0 m_1 and m_2 are the left, central and right slopes respectively.
- $\gamma(x)$ is used to approximate a non-linear expression as the union of multiple segments each one having its own slope.

Considering that $N = 2$, the FHN's phase portrait is composed of slow and fast branches and it is enough to realize a slow-fast limit cycle where the outer branches determine the stance and swing phases of each leg that can be controlled by modulating the corresponding slope parameters. If the number of segments is equal to $N + 1$, then the number of breakpoints in the PWL is N. Furthermore, the $i - th$ slope is referred to $m_i \in R$, with $i \in \{0, ..., N\}$ being m_0 the slope of the leftmost segment and m_N the slope of the rightmost one.

In order to use the FHN's neuron dynamic to control the Mini Cheetah's gait, the state variables x_1 and x_2 are normalized and used to control the joint position of the hip

and knee, respectively. The direction control of the robot is implemented by calculating the position and orientation errors between the robot's centre of mass and the next target point to be reached along the path, which is divided into a certain number of waypoints. Therefore, the error is reduced by acting on the leg trajectories, allowing the robot to steer when necessary. On the basis of the analysis performed by the authors in [26], a suitable range for the parameters m_0 and m_2 was chosen as [-5, -1]. The values adopted for the parameters are reported in Table 3.1.

Table 3.1: FHN'neuron parameters.

Parameter	**Value**
a	0.7
b	0.8
ϵ	0.01
a_0	0
a_1	-1
b_1	1
b_2	-1
m_0	[-5, -1]
m_1	1
m_2	[-5, -1]

The locomotion gaits can be generated by opportunely synchronising the phases of the four oscillators seen in Fig.3.3 connected through a master-slave scheme and controlled using the nullcline-based synchronization strategy as discussed in [24]. In Fig.3.4(a), the neural dynamics are shown taking into account three different pairs of m_0 and m_2 parameters. By varying the value of the m parameters, it is possible to have a change in the robot's gait. In fact, in Fig.3.4(b), the frontal right foot trace, as recorded by the leg foot position sensors, is reported, producing an effect in terms of energy efficiency; the adopted sampling time is 25 ms. The stepping diagram obtained from the simulated robot while walking on flat terrain with a trot gait is reported in Fig.3.4(c) ($m_0 = -1$, $m_2 = -1$, $\phi_{BR} = 0$, $\phi_{FR} = \phi_{BL} = \pi$). The stance phase is detected when the force sensor on each leg tip is above 10% of the maximum detected value. The optimization of the m parameters is therefore needed to find the best locomotion condition to minimize the energy consumption.

3.3 Matsuoka's neuron and environmental feedback

Even if CPG can work without any information coming from external stimuli, a feedback mechanism is essential in the fine tuning of legs motions. In literature, a large amount

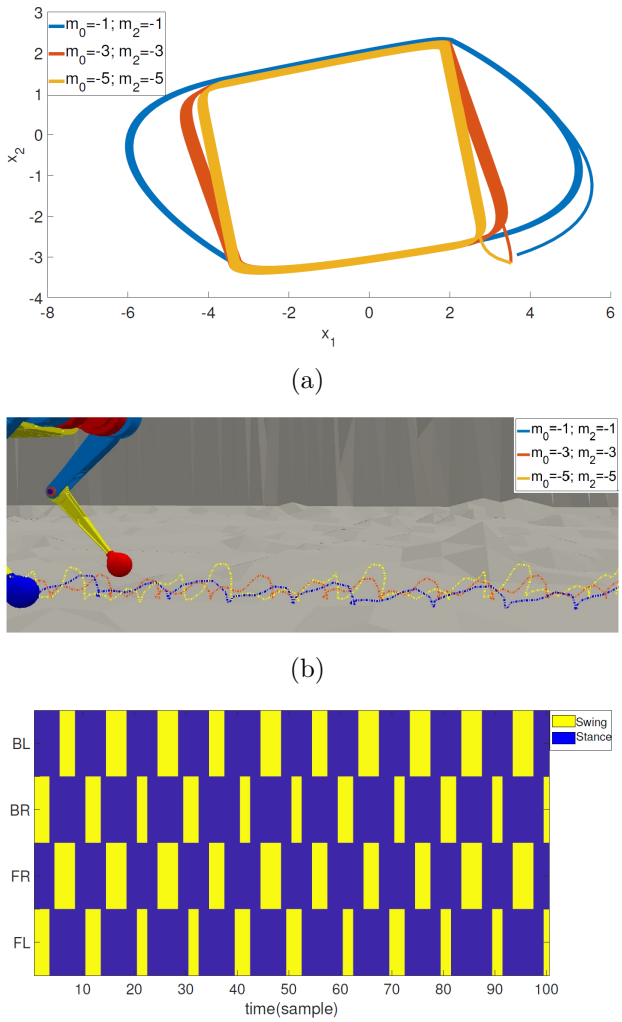


Figure 3.4: Application results of the neuron model to the robot locomotion control. (a) The FHN limit cycle shapes change based on the m parameters; (b) traces of the right front leg tip obtained for the selected m; (c) Stepping diagram obtained during the trot gait ($m_0 = m_2 = -1$).

of different strategies were introduced in this perspective. In [27] and [28], an approach in introduced, which aims at demonstrating that structured locomotion patterns can also emerge without having a pre-imposed set of basic oscillations, but, on the contrary, based uniquely on an environmental feedback. This provides similar results to the decentralised locomotion control, introduced by Holk Cruse which led to the development of the walknet

network [29].

Our neural structure is adaptive and it is based on the proprioceptive information and the exteroceptive signals acquired through ground contact sensors. The importance of feedback will be further emphasised in chapter 6, where, based on [30, 31], the Model Predictive Control (MPC) based control mechanisms generate the high-level descending commands used by the CPG environment feedback-based. A focus on the MPC control will be given in Chapter 4.

As shown in Fig.3.5, the quadruped consists of four identical legs equipped with three revolute joints. Each foot has a cylindrical shape, and it is linked to the leg through a prismatic passive joint that works as a shock absorber. The weight of the robot trunk is not uniformly distributed: there are two main parts in the front and hind of the trunk interconnected through a light-weight bar, mimicking the presence of the head, abdominal parts, and tail. In this way, the robot's weight is concentrated on the front and rear legs of the robot. The robot model was developed in CoppeliaSim.

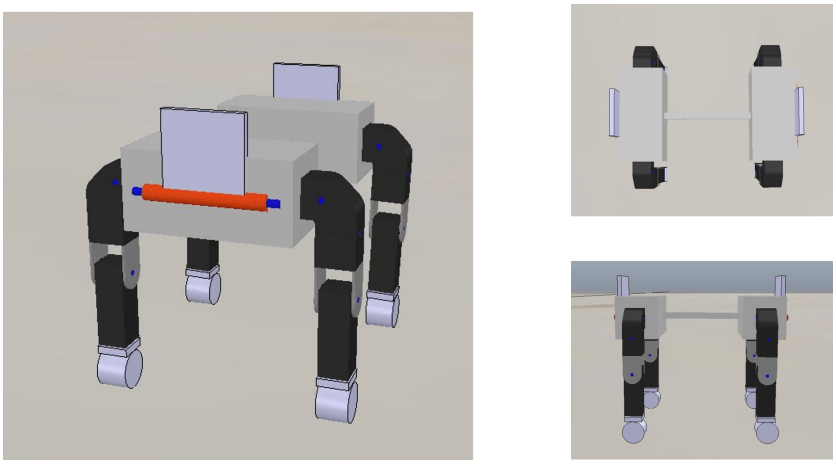


Figure 3.5: Quadruped model developed in a dynamic simulation environment.

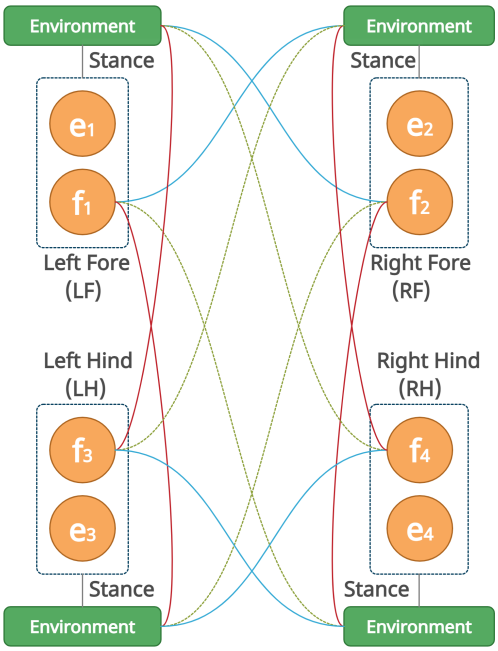
The neural control architecture is schematically reported in Fig.3.6 (a) where each leg is controlled through a half-center oscillator implemented through two interconnected dynamical systems mimicking the extensor and flexor dynamics of muscles and taking inspiration from the Matsuoka neuron [32]. A load sensor is equipped on the tip of each leg, in fact, each leg controller is not directly interfaced with the others but they are coupled through the environment.

In these equations, the suffix e, f, and i denote the extensor, the flexor, and the leg number (i.e. 1: left front, 2: right front, 3: left hind, 4: right hind), respectively.

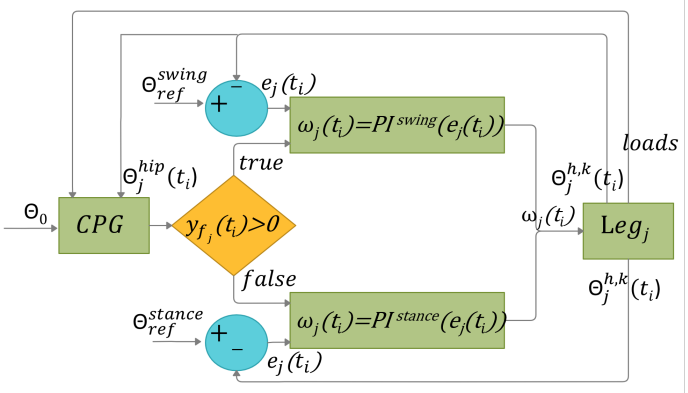
$$\dot{x}_{1_{ei}} = \epsilon_r(-x_{1_{ei}} - bx_{2_{ei}} + \gamma y_{fi} + s + feed1_{ei}), \tag{3.9a}$$

$$\dot{x}_{2_{ei}} = \epsilon_a(-x_{2_{ei}} + y_{ei}), \tag{3.9b}$$

$$y_{ei} = x_{1_{ei}} H(x_{1_{ei}}) \tag{3.9c}$$



(a)



(b)

Figure 3.6: Scheme of the locomotion control architecture. (a) The afferent load feedback connection is reported: ipsilateral connections (red line), diagonal connections (green line), and contralateral connections (blue line). (b) Feedback control scheme of each robot leg.

$$\begin{aligned}
\dot{x}_{1_{fi}} &= \epsilon_r(-x_{1_{fi}} - bx_{2_{fi}} + \gamma y_{ei} + s + feed1_{fi} \\
&\quad + feed2_{fi}), \qquad (3.10a)\\
\dot{x}_{2_{fi}} &= \epsilon_a(-x_{2_{fi}} + y_{fi}), \qquad (3.10b)\\
y_{fi} &= x_{1_{fi}} H(x_{1_{fi}}) \qquad (3.10c)
\end{aligned}$$

where $H(x)$ is the Heaviside function:

$$H(x) = \begin{cases} 0, & if \quad x < 0 \\ 1, & \text{otherwise} \end{cases}$$

The sensory information signals provided are detailed in the following.

$$\begin{aligned}
feed1_{\{e,f\}i} &= \pm\, k_1 \cdot (\theta_i - \theta_0), \qquad (3.11a)\\
feed2_f &= [feed2_{f1}, \cdots feed2_{fi}, \cdots feed2_{f4}]^T \qquad (3.11b)\\
&= K_2 \cdot L,\\
K_2 &= k_{ip} Ip + k_{co} Co + k_{di} Di \in R^{4\times 4}, \qquad (3.11c)\\
L &= [l_1, l_2, l_3, l_4]^T \in R^4 \qquad (3.11d)
\end{aligned}$$

The Ip, Co, Di parameters are matrices that are reported in Eq.3.12:

$$Ip = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} Co = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} Di = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad (3.12a)$$

The presented model is a parallel distributed dynamical system where the neurons are structured to match the leg structure. All the other parameters are tuned based on the locomotion gait to be obtained as will be schematized and discussed in the following.

- Considering all single cells of the CPG controller, each one is made up of a 4-th order nonlinear system, including two sub-units that stand for the flexor-extensor couple [33] and, in each of these, i.e. extensor and flexor, the two-state variables represent the membrane potential ($x_{1_{\{e,f\}i}}$) and the recovery variable ($x_{2_{\{e,f\}i}}$), respectively.
- The recovery variable $x_{2_{\{ef\}i}}$ inhibits the membrane potential $x_{1_{\{ef\}i}}$ through the constant parameter b.
- ϵ_r and ϵ_a are the time constants used to determine the basic frequency of the CPG and the bias term s models the descending signal from the higher brain centres.

- Within each half-center CPG, the parameter γ weights the flexor/extensor interaction and each hip joint angle is multiplied by the constant gain k_1 and provided in input to the half-center CPG of each leg [33], as in Eq. 3.9 and 3.10. If this feedback would be eliminated walking would be impossible because of the loss of the CPG rhythm during the stepping by the model.
- The terms $feed1_{\{e,f\}i}$ and $feed2_{\{e,f\}i}$ represent the sensory feedback, in particular:
  - $feed1_{\{e,f\}i}$ indicates the error between a reference angle θ_0 and the actual hip joint angle θ_i (plus or minus signs are referred to the extensor and flexor, respectively).
  - $feed2_{\{e,f\}i}$ weights the afferent loads, represented by the vector $L \in R^4$, from the neighbouring legs. This varies according to the values of the following gains acting on the legs: k_{ip} (ipsilateral), k_{co} (contralateral) and k_{di} (diagonal) that depends on the used gaits.
- $y_{\{e,f\}i}$ are the outputs of the extensor and flexor neurons of the i-th leg; these are discontinuous and non-negative terms due to the Heaviside function $H(\cdot)$.

All the above parameters are the same for each half-center CPG (i.e. for each leg).

Considering Fig.3.6(b), position control is applied to each leg. It is based on the output from its half-center CPG, providing information related to the stance or swing phase of the leg. The control flow can be schematized into three subparts:

1. The first is the phase generation in which, on the basis of the actual sensory information coming from the robot legs, the half-center CPGs output, (i.e. y_f and y_e), are computed.
2. The second phase is the parameters selection where, based on the flexor output, a PI controller is selected. In particular:
   - If $y_f(t) > 0$ then the leg is led to the swing phase adopting the set of swing PI parameters, that are K_P^{swing} and K_I^{swing}, used to reach a goal swing position θ_{ref}^{swing}.
   - On the contrary, the leg is led to the stance phase through the set of stance PI parameters, i.e. K_P^{stance} and K_I^{stance}, adopted to reach a target stance position θ_{ref}^{stance}.
3. The last phase is the actuation in which, after selecting the PI parameters and the target position, these are used in the PI controller of each leg providing an angular speed to actuate the joints.

In particular, the PI controller equation of the j-th leg is:

$$\omega_j(t) = K_{P_j}^{\{sw,st\}} e_j(t) + K_{I_j}^{\{sw,st\}} \int e_j(t_i)\, dt, \tag{3.13a}$$

$$e_j(t) = \theta_{ref}^{\{sw,st\}} - \theta_j^h(t) \tag{3.13b}$$

where:

- $\omega_j(t)$ is the output angular velocity.
- $\theta_j^h(t)$ is the current angle.
- $e_j(t_i)$ is the angular position error and they are referred to the jth leg.
- $\theta_{ref}^{\{sw,st\}}$ are vectors containing the hip and knee target joint angles during the swing/stance phase of the j-th leg.
- $K_{P_j}^{\{sw,st\}}$ and $K_{I_j}^{\{sw,st\}}$ are the proportional and the integral gains in the swing/stance phase of the j-th leg, respectively.
- $\theta_{ref}^{sw,knee}$ is the angle position to which each leg retracts during the swing phase to avoid tripping when the hip moves towards the target joint angle $\theta_{ref}^{sw,hip}$.
- Considering the knee, this is extended until it reaches a target angle $\theta_{ref}^{st,knee}$ when the leg stance phase happens, and, in the meantime, the hip swings backwards towards a target joint angle $\theta_{ref}^{st,hip}$. This motion sequence leads the robot to complete a stride cycle and to generate forward propulsion.

PI tuning was carried out using the guidelines in [34]. The parameters used are reported in Table 3.2 and a set of PI parameters are imposed by the leg controller switching periodically between stance and swing phase on the basis of the sign of the output variable of the flexor neuron in the half-center oscillator associated with the corresponding leg.

Table 3.2: Coefficients for the low-level leg controller based on a PI. The desired positions for both stance and swing phases are reported.

Parameter	Value
K_P^{stance}	14.33
K_I^{stance}	10
K_P^{swing}	16.64
K_I^{swing}	16
$\theta_{ref}^{st,hip}$	-0.35 rad
$\theta_{ref}^{st,knee}$	0 rad
$\theta_{ref}^{sw,hip}$	1.05 rad
$\theta_{ref}^{sw,knee}$	-1.22 rad

Starting from a predefined set of parameters shown in Table 3.3, this configuration allows to generate gaits and to migrate between them by changing a subset of parameters reported in the table.

To implement a heading control, a third input signal is adopted. Specifically, the input is added to Eq. 3.10a and consists in:

Table 3.3: Parameters adopted in the CPG structure.

Parameters	**Gait**			
	Lateral Sequence	Trot	Canter	Gallop
ϵ_a	1.67			
b	3			
γ	2			
θ_0	0			
k_1	3			
s	2.2	2.6	3	3
ϵ_r	6.25	8.33	16.67	16.67
k_{ip}	-0.04	0	0.08	0.08
k_{co}	0.04	0.04	-0.04	-0.04
k_{di}	0	0.08	0	0

$$feed3_{fi} = c_i \cdot S_c, \qquad i = 1, \ldots, 4 \tag{3.14}$$

where S_c is the steering command and c_i is the i^{th} element of a template vector defined as:

$$C = \begin{bmatrix} +1 \\ -1 \\ +1 \\ -1 \end{bmatrix} \tag{3.15}$$

that regulates the contribution of the steering command among the legs. By acting on S_c, a control system can allow the robot to steer and, therefore, follow an imposed trajectory.

3.4 SO(2)-networks as neural oscillators

In [35], a CPG control composed of four identical and decoupled neural SO(2) oscillators [36] is presented. In particular, a single leg of the quadruped is controlled by the decoupled CPG consisting of two fully connected standard additive time-discrete neurons both using a sigmoid transfer function. The discrete-time dynamics of two neuron networks with standard additive neurons is presented in the following. In general, it is given by a 6-parameter family of maps $f_\rho : R^2 \to R^2$, $\rho = (\theta_1, w_{12}, w_{11}, \theta_2, w_{21}, w_{22}) \in R^6$, where θ_i denotes the bias term of neuron i, and w_{ij} the synaptic weight from neuron j to neuron i. The output of a neuron is, in general, given by a sigmoidal transfer function σ, which here is chosen to be the hyperbolic tangent $\sigma = tanh$. Considering, for convenience, $\theta_1 = \theta_2 = 0$, the resulting two neuron dynamics are given by the following equations

$$a_1(t+1) := w_{11}\sigma(a_1(t)) + w_{12}\sigma(a_2(t)), \tag{3.16a}$$
$$a_2(t+1) := w_{21}\sigma(a_1(t)) + w_{22}\sigma(a_2(t)). \tag{3.16b}$$

where a_i denotes the activity of neuron i.

Having identified the Jacobian $Df_\rho(0)$ at the origin with the weight matrix w of the network, it is now easy to construct networks which correspond to configurations guaranteeing that the origin as a fixed point attractor undergoes a Neimark-Sacker bifurcation. Such networks have a weight matrix w satisfying $\det w = 1$. A special type of matrices, satisfying this condition, are the elements of the special orthogonal group SO(2). They are associated with rotations in the plane and a standard representation of these elements is given in terms of $sin(\varphi)$ and $cos(\varphi)$ of the rotation angle φ. Thus, convenient weight matrices are of the form

$$w = Df_\varphi(0) = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} cos(\varphi) & sin(\varphi) \\ -sin(\varphi) & cos(\varphi) \end{pmatrix} \tag{3.17a}$$

so that it is possible to consider the dynamics of Eq. 3.16 as a one parameter family of maps with parameter -$\pi \leq \varphi \leq \pi$, such that

$$a_1(t+1) := cos(\varphi) \cdot tanh(a_1(t)) + sin(\varphi) \cdot tanh(a_2(t)), \tag{3.18a}$$
$$a_2(t+1) := -sin(\varphi) \cdot tanh(a_1(t)) + cos(\varphi) \cdot tanh(a_2(t)). \tag{3.18b}$$

Because here one wants to obtain almost sinusoidal output signals from the network, quasi-periodic attractors are preferred. Considering $\hat{D}$ the determinant of $Df_\varphi(0)$, therefore one has to go slightly beyond the line $\hat{D} = 1$ crossing the Neimark-Sacker bifurcation set. To do this one may introduce a second parameter $\alpha > 1$ to obtain $\hat{D} > 1$ for the determinant of the Jacobian $Df_\varphi(0)$, and the weight matrix of such a network is given by

$$w = Df_{(\alpha,\varphi)}(0) = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \alpha \cdot \begin{pmatrix} cos(\varphi) & sin(\varphi) \\ -sin(\varphi) & cos(\varphi) \end{pmatrix} \tag{3.19a}$$

and the eigenvalues $\lambda_{1,2}$ of the Jacobian satisfy $\lambda_{1,2} = \alpha \cdot e^{\pm i \cdot \varphi}$. For obvious reasons, networks with a weight matrix of the type of Eq. 3.19 will be called *SO(2)-networks*.

3.5 Conclusion

In this chapter, different CPG approaches for quadruped locomotion have been shown considering different structures: reaction-diffusion system, FitzHugh–Nagumo's neuron,

Matsuoka's neuron and environmental feedback and SO(2)-networks as neural oscillators. In the first case, a reaction-diffusion system, outlining the Laplacian operator, is implemented. In the second case, the locomotion is controlled using the dynamics generated using a FitzHugh–Nagumo's neuron considering different parameter settings. In the third case, the CPG is implemented using oscillators synchronized through an environmental feedback system based on the proprioceptive information and the exteroceptive signals acquired through ground contact sensors. In the last structure, the CPG control is composed of four identical and decoupled neural SO(2) oscillators.

Chapter 4

Liquid State Machine for the Ground Reaction Forces prediction[1]

4.1 Recurrent Neural Networks

Before addressing the issue of Spiking Neural Networks, a reference to the Recursive Neural Networks from which the previous ones derive is a must. In the field of machine learning and artificial intelligence, the Recursive Neural Networks (RNNs) represent a large class of computational models biologically inspired by brain modules. A RNN is a type of artificial neural network including neurons connected in a loop so that the output values of a layer of a higher level are used in the input of a layer of a lower level. These deep learning algorithms are commonly used for explicity time depending problems so that the input data are encoded as time series. A peculiar characteristics of RNN is memory: they naturally exploit information from prior inputs to influence the current input and output [37, 38]. In Fig.4.1 the difference in information flow between a RNN and a Feed-forward Neural Network (FNN) is illustrated. As it is possible to see, RNN has a recurrent connection on the hidden state. This looping constraint ensures that time dependent information is captured in the input data.

[1]The results reported in this chapter are extracted from "Ground Reaction Force Estimation in a Quadruped Robot via Liquid State Networks - Paolo Arena, Maria Francesca Pia Cusimano, Luca Patanè, Lorenzo Emmanuele Meli, Poramate Manoonpong and Salvatore Taffara - 2022 International Joint Conference on Neural Networks (IJCNN)" and "Insect-Inspired Spiking Neural Controllers for Adaptive Behaviors in Bio-Robots - 2022 IEEE Instrumentation and Measurement Magazine - Paolo Arena, Alessia Li Noce, Luca Patanè, and Salvatore Taffara".

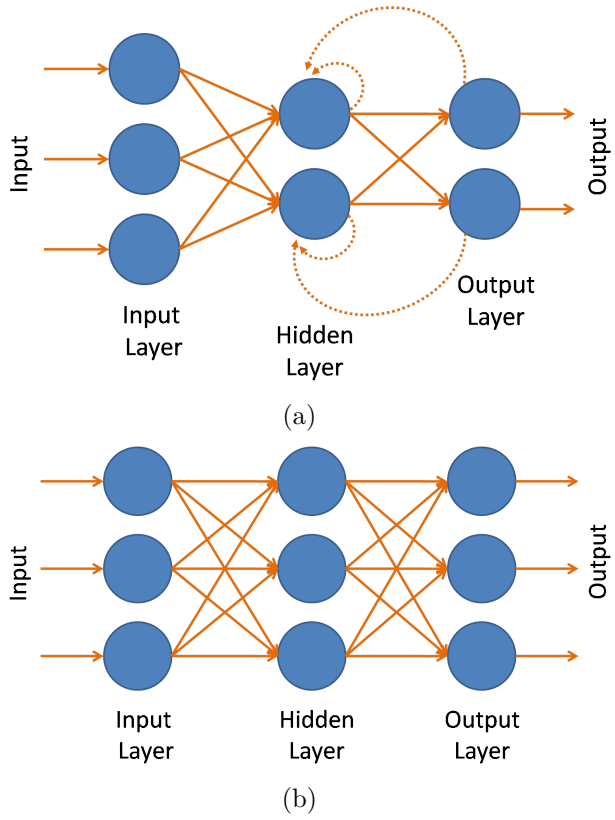


Figure 4.1: Comparison of RNN (a) and FNN (b).

4.1.1 Mathematical background

Let a problem of learning a functional relation between a given input $\mathbf{u}(n) \in R^{N_u}$ and a desired output $\mathbf{y}_{target} \in R^{N_y}$, where $n = 1, ..., T$, and T is the number of data points in the training dataset:

A *non-temporal* task involves learning a function

$$\mathbf{y}(n) = \mathbf{y}(\mathbf{u}(n)) \tag{4.1}$$

such that the error measure $E(\mathbf{y}, \mathbf{y}_{target})$ is minimized, for instance through, the NRMSE.

A *temporal task* is obtained when u and y_{target} are signals defined in a discrete time domain $n = 1, ..., T$ and the goal is to learn a function:

$$\mathbf{y}(n) = y(..., \mathbf{u}(n-1), \mathbf{u}(n)) \tag{4.2}$$

such that the error $E(\mathbf{y}, \mathbf{y}_{target})$ is minimized.

The difference between the temporal and non-temporal task is that function to learn $y(\cdot)$ works as a memory in the first case while it is memoryless in the second one. In both cases, the underlying assumption is that the functional dependence to learn already exists in the data. For the temporal case this defines the way data stick to an additive noise model:

$$\mathbf{y}_{target}(n) = y_{target}(..., \mathbf{u}(n-1), \mathbf{u}(n) + \theta(n)) \tag{4.3}$$

where $y_{target}(\cdot)$ is the relation to be learned by $y(\cdot)$ and $\theta(n) \in R^{N_y}$ is a noise term limiting the learning precision in matching the learned $\mathbf{y}_n$ and the $\mathbf{y}_{target}$.

It comes that the error $E(\mathbf{y}, \mathbf{y}_{target})$ is small when the task is learned with good accuracy or precision. Normally one part of the T data points is used for training the model and another part, unseen during the training, for testing it. Also, n, denoting the discrete time, will often be used omitting its range $1, ..., T$.

A series of drawbacks can be considered for the RNNs:

- RNNs, due to their complexity and recurrent nature, show a slow computation and difficulty in the train process [39].
- RNNs suffer from the problem of vanishing gradients, which hampers the learning of long data sequences. The gradients carry information used in the RNN parameter update and when the gradient becomes smaller and smaller, the parameter updates become insignificant which means no real learning is done [40, 41].

4.2 Reservoir computing: Liquid State Machines and Echo State Networks

To fix the problems above, in 2002, a new approach to RNNs design and training was proposed by Maass et al. that proposed the Reservoir Computing (RC) theory [42, 43]. In this direction, Echo State Networks (ESNs) and Liquid State Machines (LSMs) are particular RNNs randomly generated where only the readout layer is trained. LSM make use of Spiking Neural Networks (SNNs) [44, 45].

RC is a computational framework suited for temporal/sequential data processing that is derived from RNNs. A reservoir computing system consists of a reservoir for mapping inputs into a high-dimensional space and a readout for pattern analysis from the high-dimensional states in the reservoir. The reservoir is fixed and only the readout is trained with a simple method such as linear regression and classification. Due to the low training cost, the most important advantage of RC with respect to RNNs is the fast learning. Another advantage is that the reservoir without adaptive updating is amenable to hardware implementation using a variety of physical systems, substrates, and devices.

The main difference between ESNs and LSMs is that the last ones are biologically inspired spiking neural networks, whereas the first ones are rate-based approximations. ESNs were originated as a Machine Learning algorithm for classifying/forecasting time series, while LSMs were designed with the goal of modeling and understanding biological neural networks [46].

4.2.1 Examples of LSMs and ESNs applications

In [47], a study of the cerebellum as an LSM is carried out. The granular layer acts as the reservoir, while the Purkinje cells act as the readout neurons. Soures et al., in [46], use LSM for video activity recognition using the DogCentric dataset consisting of 209 videos recorded for ten different activities being performed by four different dogs from a first-person view point. In [48], Lazar, Pipa and Triesch analyse the temporal patterns problem in time series, in particular, they combine spike timing dependent plasticity and intrinsic plasticity to maintain homeostasis of neuronal activity that stabilizes the LSM. In [49] the authors show that offline-trained LSMs implemented in the SpiNNaker neuromorphic processor are able to classify visual events.

In [50], Jaeger et al. study ESNs made of leaky integrator neurons and they present basic stability conditions, investigate parameter optimization by stochastic gradient descent, and demonstrate the usefulness of leaky integrator ESNs in test cases that require long time constants and insensitivity to time warped patterns. Steil, in [51], uses the intrinsic plasticity of neurons to propose a new local, unsupervised adaptation rule for in-reservoir connections that improves the richness of its dynamics from an information point of view. In [52], Xue, Yang and Haykin try to optimize reservoir dynamics who implement lateral inhibition structures in a modular ESN to improve the richness of the reservoir dynamics. Venayagamoorthy, in [53], applies ESNs to monitor a multi-machine power system, demonstrating improved performance with much simpler training when compared with time delay neural networks. Skowronski and Harris, in [54], bring ESNs to speech recognition and show improved performance with respect to Hidden Markov Models in low signal to noise ratio regimes. Tong et al. in [55] use ESNs in language modeling to learn grammatical structure and show that their performance is similar to that of Elman networks, even though the ESN does not train the recurrent connections. In Table 4.1, a summary of the above articles, with the related applications, is reported.

Table 4.1: Applications of LSMs and ESNs.

Reference	Type of network	Application
[47]	LSM	Modelling of the cerebellum.
[46]	LSM	Video activity recognition.
[48]	LSM	Neuronal plasticity analisys.
[49]	LSM	Classification of visual events.
[50]	ESN	Leaky integrator neurons.
[51]	ESN	Optimization of reservoirs of analog neurons.
[52]	ESN	Neural lateral inhibition.
[53]	ESN	Monitoring of a multi-machine power system.
[54]	ESN	Speech recognition.
[55]	ESN	Learn of grammatical structures.

4.3 LSM: theory fundamentals

LSMs are randomly generated SNNs in which the internal connectivity parameters remain static during the training process, acting as a reservoir. This latter is excited by input signals and has a state, a non-linear transformation of the input's history, that is connected a to linear readout unit. The state of the reservoir can be seen as a mapping of the input data into a higher dimension where the prediction or classification task is easier to solve, similar to the kernel methods like Support Vector Machines.

If set properly, LSMs can represent spatiotemporal inputs in a higher dimensional space where non-linear combinations of frequencies resonate, providing useful information that makes the characterization of the input simpler to infer, while requiring a significantly less amount of computational resources compared to a RNN trained in a standard way as the connectivity weights among layers of the network are not trained, except for the output or classification layer [56, 57].

Mathematically, a LSM maps input streams $u(\cdot)$ onto output streams $y(\cdot)$ exploiting the dynamics of the internal liquid state $x(t)$. Input signals are translated into spiking sequences that are injected into the neurons belonging to the liquid layer. The different readout maps extract the liquid state information, modulated by trainable weights. Of course, the target value $y(t)$, i.e. the desired output at time t, may depend on the values $u(s)$ of the input streams at previous time steps s. However, there is no need to explicitly introduce the input or output delayed data since memory is built internally.

The liquid layer L maps input functions $u(\cdot)$ onto a state function:

$$x(t) = L(u(t)) \tag{4.4}$$

Moreover, it is important to specify that the readout map is a memory-less function f that transforms, at every time t, the current liquid state $x(t)$ into the output:

$$y(t) = f(x(t)) \tag{4.5}$$

f is generally chosen in a task-specific way. The memory-less property refers to the fact that f does not need to retain any memory of previous states of the liquid, but as a result of learning, the readout map will contribute to the system memory [58, 59].

4.4 Ground Reaction Forces estimation using LSMs

In this section, the design of an efficient robot state estimation method based on RC is shown. The local proprioceptive information acquired at the level of the leg joints, the torques (TRQs), of a simulated quadruped robot are used as input of a LSMs to predict the GRFs, so, the aim is to create a map between TRQs and GRFs. The robot taken into account is the simulated version of the Lilibot [35], a small-sized and reconfigurable bio-inspired robot with multiple real-time sensory feedback [60, 61]. During the simulations, the Lilibot walks at a fixed speed showing a trot gait, i.e., each leg is in phase with its

diagonal and 180° out of phase with the other two legs. The quadruped is shown in Fig.4.3. The robot legs are defined as: FR (front right leg), HR (hind right leg), FL (front left leg), HL (hind left leg). The involved joints are H_1 (hip 1 joint), H_2 (hip 2 joint), K (knee joint). The target signals consist of the vertical component of the ground reaction forces at the tip of each leg. The simulations are performed in totally flat terrain.

In Fig.4.2 the control system used to control the robot movement is shown. The communication between the controller (ROS node1) and the CoppeliaSim simulator (ROS node2) is accomplished through three ROS topics and a parameter server. The topics include:

- A "motorValues" topic transmitting motor commands of joints from the controller to the simulated robot;
- A "sensorValues" topic transmitting sensory signals of the simulated robot to the controller;
- A "neuralNetworkOutputs" topic transmitting the outputs of the sub-control modules. The parameters of the controller are accessed through a ROS parameter server. The communication between the controller (ROS node1) and the real robot (ROS node3) is also performed in the same manner through the ROS topics and the parameter server [35].

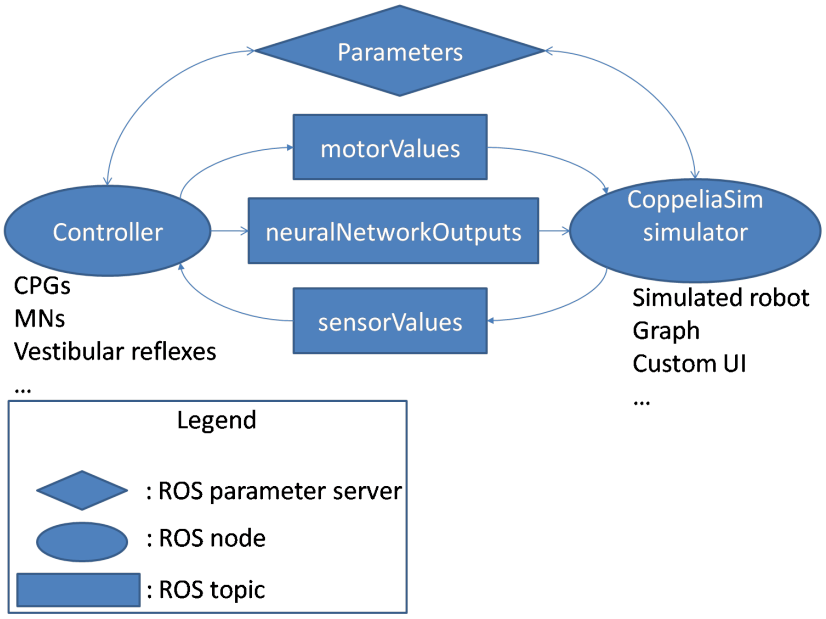


Figure 4.2: Control scheme for the Lilibot. ROS and CoppeliaSim simulator communicate together.

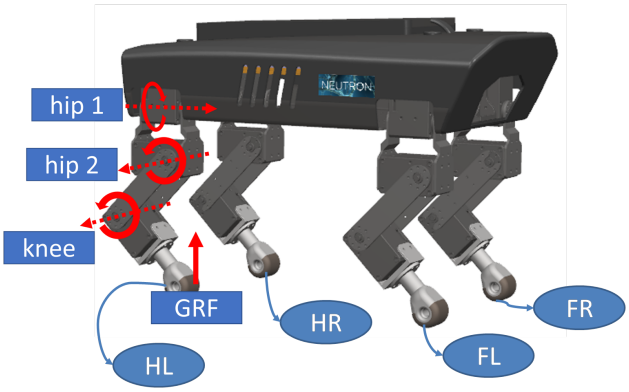


Figure 4.3: Lilibot robot used in the dynamic simulation environment. Each leg is characterized by three actuated joints for a total of 12 degrees of freedom.

The steps of the whole algorithm are schematized in Fig.4.4. The details for each block will be given in the rest of the chapter.

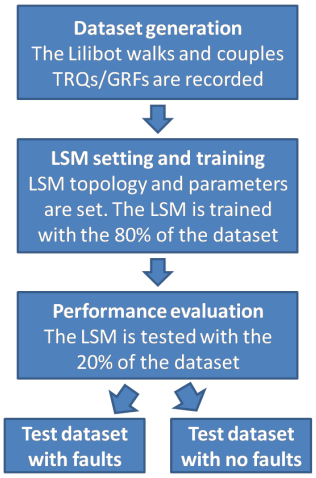


Figure 4.4: Structure of an LSM that receives in input the coded data.

4.4.1 Dataset generation

The scheme used to train and to test the LSM is shown in Fig.4.5 where the TRQs ($Hip1_{FL}$, $Hip2_{FL}$, $Knee_{FL}$, $Hip1_{FR}$, $Hip2_{FR}$, $Knee_{FR}$, $Hip1_{HL}$, $Hip2_{HL}$, $Knee_{HL}$, $Hip1_{HR}$, $Hip2_{HR}$, $Knee_{HR}$) represent the input and the GRFs (GRF_{FL}, GRF_{FR}, GRF_{HL}, GRF_{HR}) are the output of the LSM. In particular, 800 records (80% of the dataset) are used for the learning phase and 200 records (20% of the dataset) for the testing phase.

These data are collected while the simulated Lilibot walks. In particular, three datasets are generated for flat terrain, downhill, and uphill terrain with a 5 degree of slope.

TRQs						GRFs			
Hip_{1FL}	Hip_{2FL}	$Knee_{1FL}$	Hip_{1FR}	Hip_{2FR}	...	GRF_{FL}	GRF_{FR}	GRF_{HL}	GRF_{HR}
a	b	c	d	e	...	f	g	h	l
a'	b'	c'	d'	e'	...	f'	g'	h'	l'
...	...	...	...	...	...	...	...	...	...

LSM input LSM target

Figure 4.5: Dataset setting used to learn and test the LSM, the inputs are the TRQs while the outputs are the GRFs. Three datasets are generated, for the flat, uphill, and downhill terrain.

4.4.2 LSM setting and training

The LSM implemented in the work is schematized in Fig.4.6. There are two main parts, the first one in which the raw data are encoded to be sent to the LSM and the second one in which the LSM structure is shown.

- **Encoding and input layer**
  The TRQs generated from the 12 actuated joints distributed in the legs, three joints per leg, are the input sources of the LSM. These data have to be encoded and, for this reason, the raw input data are sent to a *step current generator* in which each input data is maintained for a given time window to be processed by the network. These steps have time periods set in the parameter $Stim_i$ and are normalised in amplitude scaling the signals in a range [0, 50] pA. Each current generator is connected to one *spike generator* that, in the proposed network, is implemented using a class I Izhikevich's neuron [62], which generates the spike train to be injected into the liquid lattice. The equation defining the Izhikevich's neuron is the following:

$$\begin{cases} \frac{dV_m}{dt} = 0.04V_m^2 + 5V_m + 140 - u + I \\ \frac{du}{dt} = a \cdot (b \cdot V_m - u) \end{cases} \tag{4.6}$$

  If $V_m \geq V_{th}$:

$$\begin{cases} V_m \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{4.7}$$

  where:

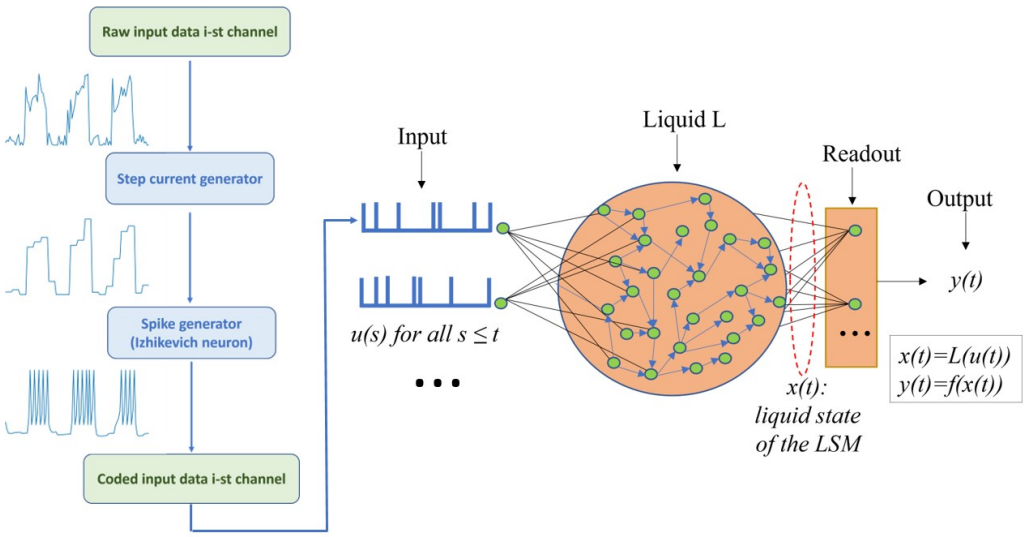


Figure 4.6: Structure of an LSM that receives in input the coded data.

- V_m is the membrane potential;
- u is the recovery variable that provides negative feedback to V_m;
- V_{th} is the spike threshold;
- I is the input current;
- a describes the time scale of the recovery variable;
- b describes the sensitivity of the recovery variable to the subthreshold oscillations of V_m;
- c represents the after-spike reset value of V_m;
- d is the after-spike reset value of the recovery variable.

The adopted values are: $a = 0.02$, $b = 0.2$, $c = -65$ and $d = 2$ to implement a Class I behaviour where the spiking frequency is proportional to the amplitude of the input current [62]. The neuron dynamic on time is shown in Fig.4.7.

Spike generators are randomly connected with a subset of lattice neurons, called *neuron targets*. The fixed-outdegree connection synaptic rule (shown in Fig.4.8) is adopted: the nodes of an input layer are randomly connected with the nodes of the following layer such that each input has a fixed out-degree N (fixed to the 10% of the number of neuron targets). The synaptic model used is the static one with uniform weight distribution, setting the minimum weight value ($W_{low} = 125$) and the maximum weight value ($W_{high} = 375$). The delay for the connections is obtained using a normal-clipped distribution, a Gaussian distribution, that takes

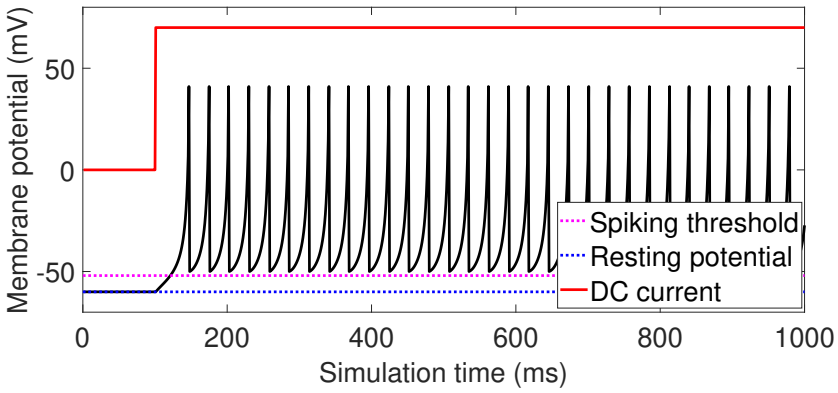


Figure 4.7: Izhikevich neuron dynamic on time.

into in the two boundary parameters low limit delay (D_{low}) and high limit delay (D_{high}) beyond the standard deviation ν and the mean σ. The adopted values are: ν=10, σ=5, $D_{low} = 3$ and $D_{high} = 200$.

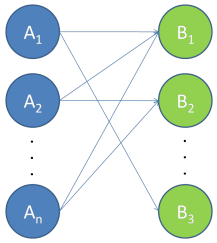


Figure 4.8: The nodes in the first layer are randomly connected with the nodes in the second layer such that each node in the first has a fixed outdegree of N.

- **Liquid layer**

  The neurons in the liquid layer are implemented with the Izhikevich's model previously introduced. The synaptic connections follow the previously adopted fixed indegree rule (Fig.4.9) with $N = 2$ for excitatory-excitatory and excitatory-inhibitory connections and $N = 1$ for the inhibitory-excitatory and inhibitory-inhibitory ones. The adopted synaptic model, introduced by Tsodyks, implements short-term synaptic depression and short-term facilitation [63]. A Gaussian distribution is used for the weight setting with $\nu = 50$ and $\sigma = 0.7 \cdot |\nu|$.

  A Poisson generator is used to inject noise inside the excitatory and inhibitory neurons. The synaptic connections are static and a normal weight distribution is adopted with $\sigma = 1$ and $\nu = 0.7$. The delay distribution is obtained using a normal-clipped structure with $\sigma = 10$, $\nu = 20$, $W_{low} = 3$ and $W_{high} = 200$.

- **Output layer**

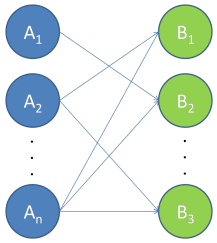


Figure 4.9: The nodes in the first layer are randomly connected with the nodes in the second one such that each node in the first has a fixed indegree of N.

To evaluate the liquid lattice activity, the exponential time decay amplitude in a window of $60ms$ for each stimulus is collected, according to the readout delay chosen. This index can be expressed with the following equation [64]:

$$f_i(t) = e^{(-\frac{t-t_i^{spike}}{\tau})} \tag{4.8}$$

where $t_i^{spike} \in [0, t]$ is the last spike time of the liquid neuron i and τ is a global fading term fixed to $20ms$ in the following simulations. The function f_i maps a spike train of a neuron i to a continuous signal. The state evolution of the LSM is then expressed as the sum, for each time window, of the exponential time decay of each readout neuron. The supervised learning rule adopted to determine the readout map is based on the Moore–Penrose inverse method as reported in the following:

$$W = (X^T X + kI_p)^{-1} X^T y_t \tag{4.9}$$

where W is the readout weight matrix, X corresponds to the state matrix that includes the state of each readout neuron for each input pattern, k is a small constant gain introduced in presence of ill-conditioned matrices, I_p is the $p \times p$ identity matrix and y_t is the target signal.

To choose the best LSM topology configuration, a grid search was performed by changing the number of excitatory and inhibitory neurons and evaluating the corresponding MSE index (see Fig4.10). The best network configuration was obtained for a LSM with a size of in total only 18 neurons having a 14-4 configuration (i.e., 14 excitatory and 4 inhibitory neurons). In this setup, all the excitatory neurons are used as readout neurons. This network is considerably small if compared with the ESN network presented in [65] where 100 reservoir neurons are considered. Inputs are coded as step currents eliciting 12 Izhikevich's class I neurons (i.e., one for each input) whose operative input range varies in $[0 - 50]pA$. This last interval was assumed as a normalization range for the input encoding and, as consequence, the "zero torques" were set equal to $25pA$. In the following analysis, an LSM with 14 excitatory neurons and 4 inhibitory neurons in the liquid layer was considered (see Fig.4.11).

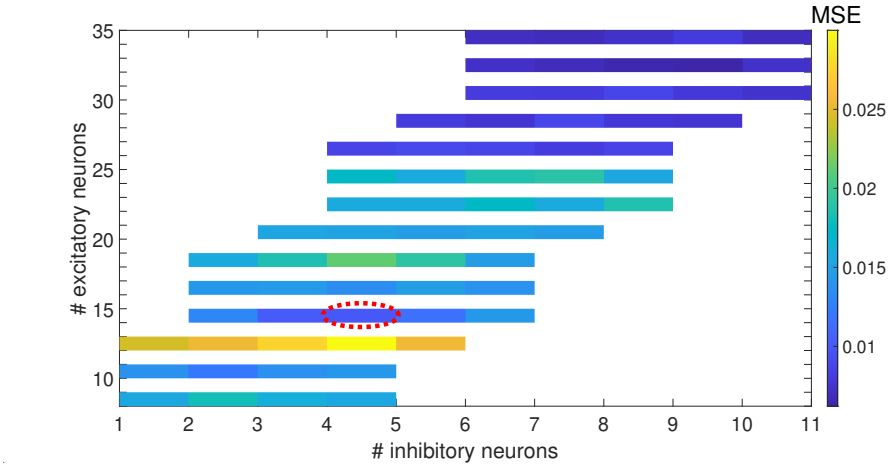


Figure 4.10: MSE values for different network topologies. The selected network is composed of 14 excitatory neurons and 4 inhibitory ones.

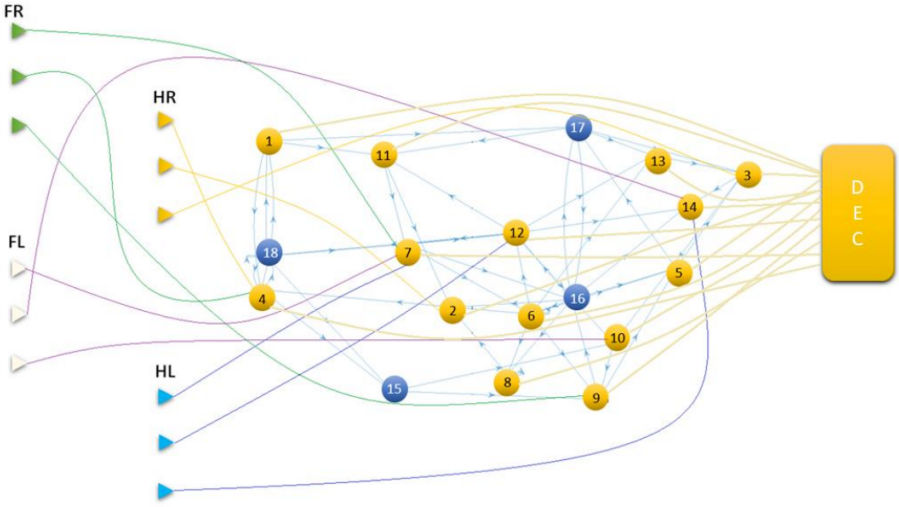


Figure 4.11: LSM 14-4 structure graph where all 14 excitatory neurons are also used as readout neurons. The inputs are represented by four groups of three triangles indicating the three joints (i.e., hip 1, hip 2, and knee) of each leg. The 14 yellow nodes represent the excitatory neurons and the 4 blue nodes are the inhibitory ones of the liquid layer. The decoder (i.e., DEC) represents the readout map which finally provides the four estimated GRFs.

A first analysis was focused on the excitatory spiking rate in order to investigate the

dynamic range for each Izhikevich's spiking neuron involved. The reservoir shows multiple distinct dynamic behaviours that underline the LSM activity in response to the injected stimulus. Some of the liquid neurons achieve a relevant spiking frequency, while some others remain silent, failing to reach the activation threshold as can be seen in Fig.4.12 where the spiking rate map related to the excitatory neurons is shown.

After the setting of the LSM is completed, this latter is trained using the 80% of the dataset TRQs-GRFs.

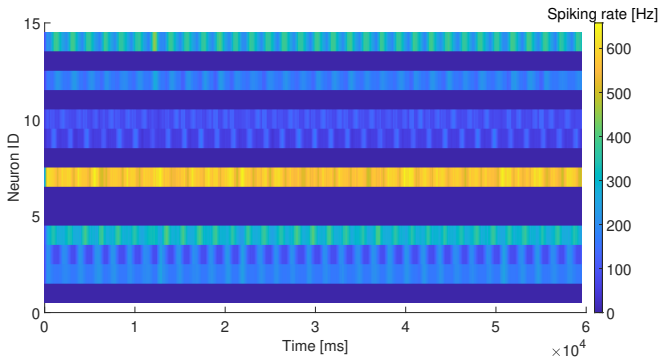


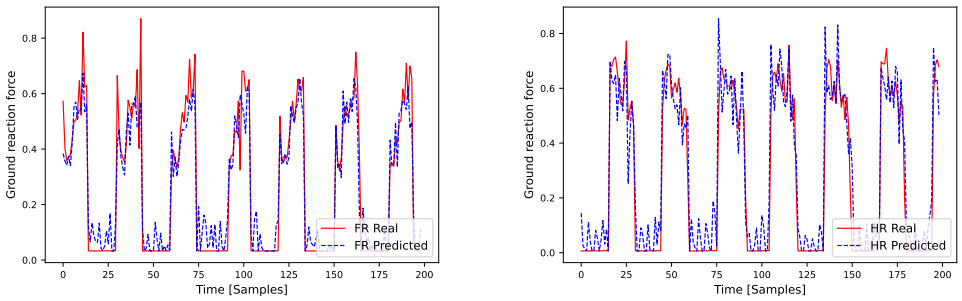
Figure 4.12: Spiking rate map showing the spiking rate for the 14 excitatory neurons. Some of the neurons (ID 1-5-6-8-11-13, dark blue) are silent during the reported simulation.

4.4.3 Performance evaluation: intact and faulty sensors

Two different types of simulations were carried out. In the first one, there is a perfectly working acquisition system and all the legs and motors work properly. In Fig.4.13 the GRFs predicted and the real ones are compared for the flat terrain case. In the second type of simulation, the presence of faulty sensors at the joint level is simulated. This is carried out by associating a "zero torque" signal to a broken joint or leg, seen as a collection of joints, as needed.

The real and estimated signals, when a fault in the time window between 200 and 800 samples is applied, are shown for the two legs FR and FL in Fig.4.14. Also, in this case, the sensory fault in FR is properly handled by the network through its fading memory and capability to exploit the available information coming from the other joints. It has to be underlined that the LSM here evaluated has been trained without any fault in the dataset. So, the network is asked to reconstruct the missing information in one or multiple joints from the data coming from all the others. Each experiment aims to exploit the dynamical response of the network in presence of unexpected failures, for different configurations. The purpose is to assess the robustness of the LSM and its capability to work as a fading memory.

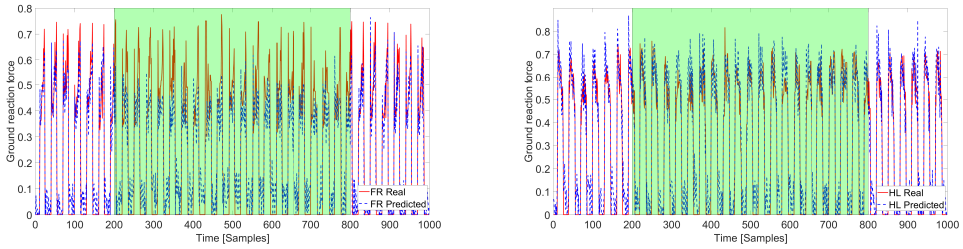
In Fig.4.15 the MSE and correlation coefficient between the estimated and actual GRFs are considered for each of the four legs when all the FR torque sensors are in fault.



(a) (b)

(c) (d)

Figure 4.13: Test results for the flat scenario predicted vs target GRFs. (a) FR leg, (b) HR leg, (c) FL leg, (d) HL leg.

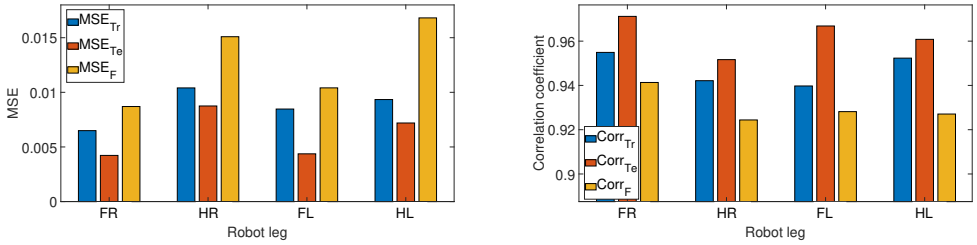


(a) (b)

Figure 4.14: Test with a fault in the sensors of all the three joints in the FR leg, occurring in the time window between 200 and 800 samples (i.e., green area). Comparison between the target and the estimated GRF signals for (a) FR and (b) the HL.

The degradation of the performance is particularly evident in the estimation of the GRF

signal for the faulty leg, however, the still high level of correlation, with a maximum degradation of about 5.7%, guarantees a good following of the stepping sequence.



(a) (b)

Figure 4.15: FR sensory system in fault: (a) MSE between the estimated and actual GRF signals and (b) correlation coefficient in training (Tr), test (Te), and in presence of faults (F).

Estimation performances in terms of MSE between the actual and estimated GRF are reported in Table 4.2 where the outcome when each leg sensory system is in fault, is reported.

Table 4.2: GRFs MSE results in test when all the joints of one leg are in fault.

	MSE			
	FR	**HR**	**FL**	**HL**
FR in fault	0.015	0.011	0.011	0.007
HR in fault	0.014	0.040	0.008	0.028
FL in fault	0.014	0.016	0.024	0.013
HL in fault	0.008	0.015	0.010	0.010

4.5 Online GRFs estimation for terrain classification

The estimation capabilities of LSM can be exploited for obtaining information on the terrain walked by the robot. In fact, even if the main role of the network is the pure prediction of GRFs from joint TRQs, the fundamental rule that the network should have learned would be the load distribution among the legs, strictly related to the type of terrain walked. In Fig.4.16, the GRFs of the HR leg are reported when the robot walks on terrains with different slopes, in particular a 5 degrees downhill and uphill. As it can be verified, even if the network was trained only with flat terrain, the network underestimates (overestimates) loads when walking uphill (downhill).

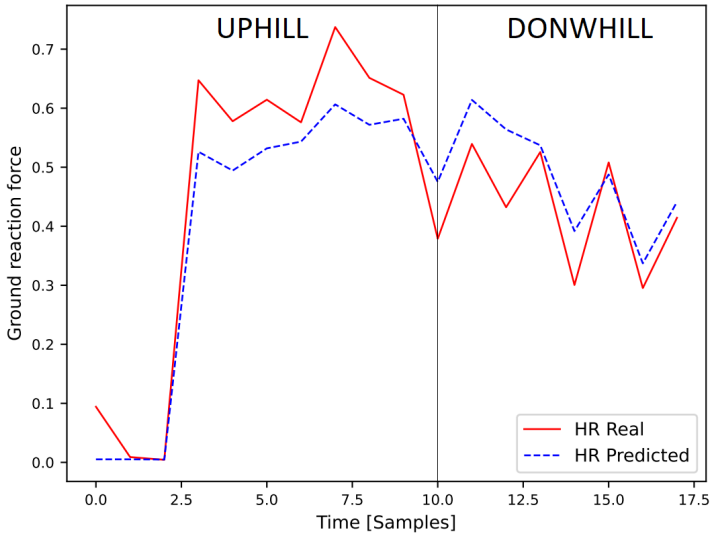


Figure 4.16: Test results in the case of uphill and downhill for the HR leg.

In order to implement an online terrain classifier, a LSM capable to estimate the GRFs during the robot walk is developed. The scheme of communication between CoppeliaSim and Nest is presented in Fig.4.17. Starting from the neural signals generated from the CPG, sequential datasets of 160 TRQs related to the 8 motors (i.e. sequential datasets of 160 rows $\times$ 8 columns representing the LSM inputs) and 160 GRFs related to the 4 legs (i.e. sequential datasets of 160 rows $\times$ 4 columns representing the LSM targets) are collected in CoppeliaSim and sent to the LSM implemented in Nest. In Nest the errors between the predicted and the target GRFs collected are obtained, generating a sequence of samples representing how much the predicted values deviate from the desired ones.

Considering the sequential errors related to the HR leg, taken as example, Fig.4.18 is obtained in which the robot is let to walk and to follow the path presented in Fig.4.19. This latter is composed of five parts: a 20° downhill, a 10° downhill, a flat part, a 10°

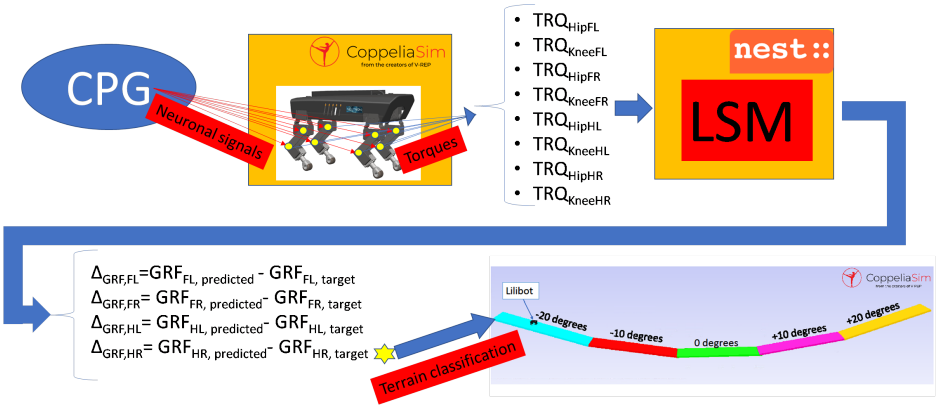


Figure 4.17: The communication scheme between the CoppeliaSim and Nest simulators. Starting from the the CPG signals sent to the quadruped motors and exploiting the LSM classification capabilities, it is possible to classify the type of terrain walked by the Lilibot.

uphill and a 20° uphill. As it is possible to verify, fixing five thresholds for the HR network errors, it is possible to obtain a terrain classifier. Table 4.3 reports the threshold values considering also the maximum and the minimum values defining each area.

Table 4.3: GRF errors thresholds, maximum and minimum values for each terrain related to the HR leg.

Terrain	Mean value	Maximum value	Minimum value
Downhill 20 degrees	0.1130	0.1523	0.0745
Downhill 10 degrees	0.0802	0.1155	0.0413
Flat	0.0215	0.0442	-0.0160
Uphill 10 degrees	-0.0237	0.0129	-0.0532
Uphill 20 degrees	-0.0427	-0.0134	-0.0729

4.6 Conclusions

In this chapter, a new architecture based on reservoir computing applied to a quadruped robot is reported. In particular, the project focuses on the creation of an efficient and robust robot state estimation method implementing a robot-environment interaction. The goal is the realization of an estimator dealing with proprioceptive and exteroceptive information, able to estimate the GRF at the tip of the leg using joint torque information acquired on a simulated quadruped robot walking on different terrain slopes. In order to test the learning capability and robustness of LSMs, unexpected faults during the tests

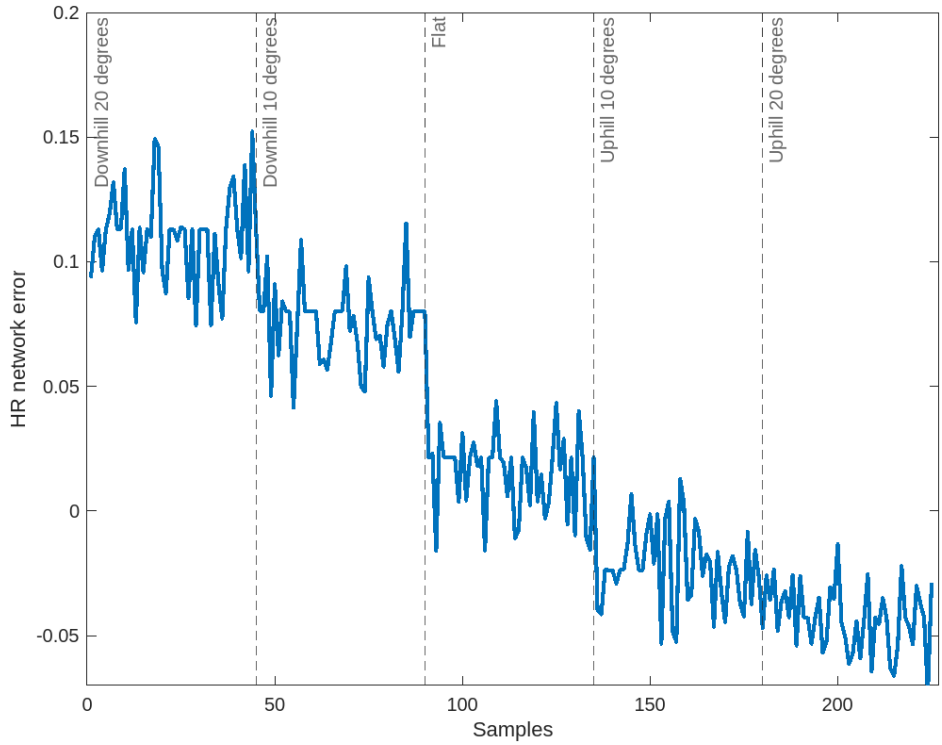


Figure 4.18: Sequential GRF errors related to the HR leg obtained considering sequential datasets of dimension 160×8 that are the LSM input and 160×4 that are the LSM output.

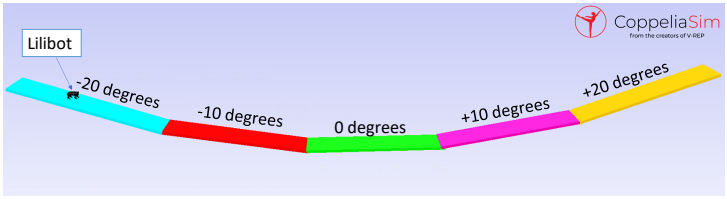


Figure 4.19: Path followed by the robot composed of: 20° downhill, 10° downhill, flat part, 10° uphill and 20° uphill.

were considered, exploiting the presence of an internal temporal memory through the recurrent connections and the potentiality of spiking neurons processing. The comparison with ESN suggests that spiking neurons in the LSM can encode temporal information more effectively than non-spiking neurons in the ESN. Exploiting the capabilities of the LSM to estimate the amplitude of the GRF, and considering such quantity strictly correlated with the terrain slope, an online terrain classifier was developed, where the robot is able to recognise the slope of the terrain walked. At this aim, a LSM capable to estimate the GRFs during the robot walk is developed building a communication between the CoppeliaSim and Nest simulator. Evidences are given related to the fact that, fixing some thresholds for the GRFs estimation errors related to the HR leg, it is possible to obtain a terrain classifier.

Chapter 5

Evaluation of robot energy consumption[1]

Considering a set of mobile robots available to accomplish a task, the best one should be chosen based on its capability. In general, not only the shorter time taken by the particular robot to travel the route should be taken into account, but also its energy efficiency in order to reduce the energy cost needed to carry out the task.

There may be energy limits that a robot, especially during climbing and walking on complex terrains, must face and an estimation of the battery life is not trivial. So it is clear that a fundamental aspect of quadrupedal navigation is the evaluation of energy consumption.

5.1 COT index

In order to evaluate the energy cost spend by a mobile robot, the Cost Of Transport (COT) index is commonly adopted in literature to express the energy efficiency in animals, mobile robots, and especially quadruped robots [20]. It is an extremely useful tool when used to indicate the effort needed by a mobile robot in accomplishing a task, for instance in travelling on a given path. The power consumption, speed, and weight (which also includes the payload) are part of the COT formula. The COT is specifically used to account for an animal or robot efficiency in relation to its capability to move, it is not able to directly express either the capability of a robot to change its speed due to its inertia or the operation cost needed to maintain a robot in standby because if the speed is zero the index goes to infinity [66, 67]. The COT is expressed as follows:

$$COT = \frac{P}{mgv} \tag{5.1}$$

[1]The results reported in this chapter are extracted from "Energy Efficiency of a Quadruped Robot with Neuro-Inspired Control in Complex Environments - Paolo Arena and Luca Patanè and Salvatore Taffara - 2022 Energies".

where P is the power consumption, m is the mass, g is the gravity acceleration and v is the velocity.

In Fig.5.1 the characteristics of the most relevant legged structures (hexapods, quadrupeds, bipeds, and monopods), regarding their COT values [68], are reported, in particular, the distribution of the COT as a function of the weight for different robots is considered. The COT indexes are evaluated in different experimental conditions that are specific for each robot. In the COT calculation, the speed is evaluated by measuring the travelled space in a given time window, therefore, representing an average speed. The red bullet refers to the average COT drawn by the simulations performed using the Mini Cheetah analyzing the COT for energy-efficient and low-speed gaits focalised on uneven terrains. It is clear that the lower the speed, the higher the cost, since, in principle, a standing-up legged robot with zero speed, has a non-zero power consumption to maintain the resting pose.

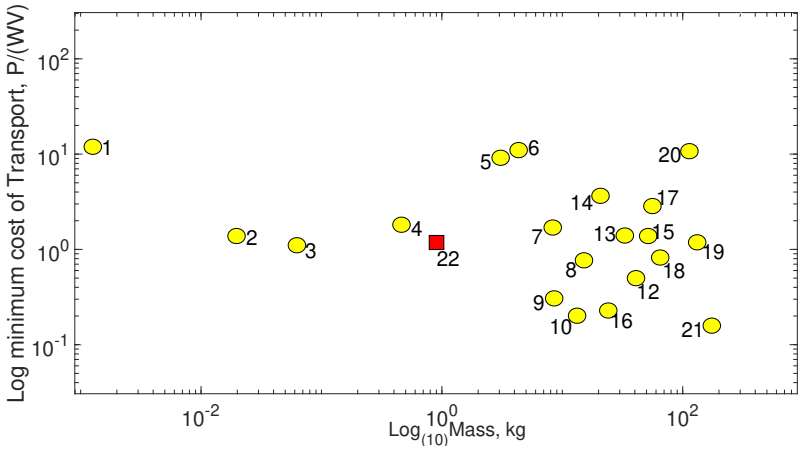
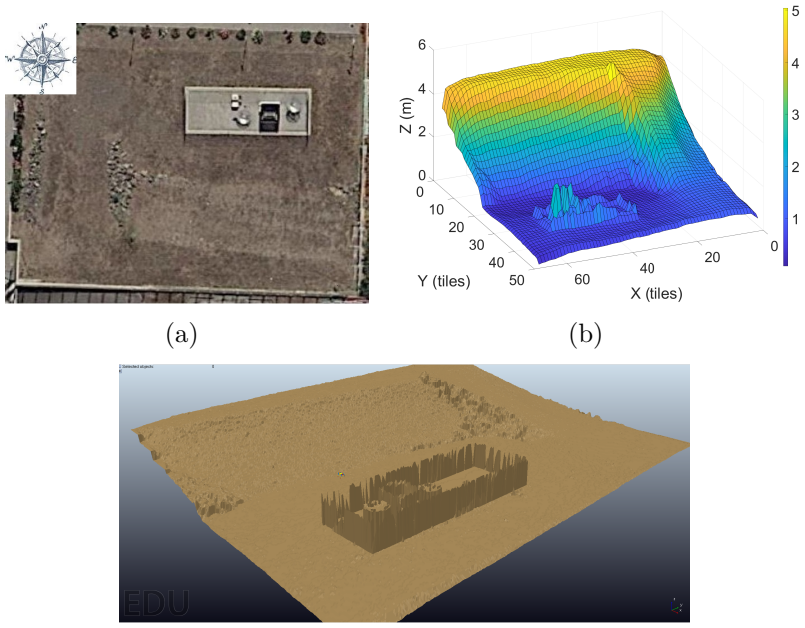


Figure 5.1: Comparison between monopod, biped, quadruped, and hexapod robots in terms of COT. The reported robots are the following: 1 HAMR-VP; 2 X2-VelociROACH; 3 DASH; 4 iSprawl; 5 Cheetah Cub; 6 MIT Learning Biped; 7 Rhex hexapod; 8 RHex-biped; 9 Cornell Ranger; 10 Cornell Biped; 11 ARL Monopod I; 12 ARL Monopod II; 13 Scout II; 14 StarlETH; 15 Fastrunner; 16 MIT Cheetah; 17 ATRIAS 2.1; 18 Asimo; 19 KOLT; 20 Big Dog; 21 ETH Cargo; 22 Mini Cheetah (Simulated).

The Mini Cheetah simulated robots used in the following tests embed the CPG based on the FitzHugh–Nagumo's neuron shown in Chapter 3. The performance analysis of the legged robot has been conducted on a simulated terrain obtained by acquiring the height map of a target site using drones [69]. The image embeds georeferencing information. Fig.5.2(a) shows an aerial image of the terrain, together with its 3D reconstruction (Fig.5.2(b)) and the appearance of the terrain in the dynamic simulator (Fig.5.2(c)).

To compare the COT performance given by the Mini Cheetah with other robotic structures, the Robotnik and the Asguard robots are taken into account (see Fig.5.3). Table 5.1 contains information related to these three robot characteristics, Table 5.2 reports the



(a) (b)

(c)

Figure 5.2: (a) Aerial image of the terrain used in simulation; (b) 3D terrain reconstruction; (c) terrain reconstructed in the dynamic simulation environment.

COT values for the Robotnik and Asguard while Fig.5.4 shows the three different areas extracted from the target terrains used to test the robot performances.

Table 5.1: Characteristics of the considered robots (h: height, w: width, d: depth).

Robots	**Weight** $[Kg]$	**Type**	**Dimension (h, w, d)** $[cm^3]$	**Clearance** $[cm]$
Mini Cheetah	9	Quadruped	$35 \times 20 \times 30$	(12-15)
Robotnik	40	Wheeled	$39.2 \times 61.3 \times 72$	18
Asguard	10	Hybrid	$28 \times 28.1 \times 33.4$	10

The highest COT values are given by the Robotnik robot due to the high powers produced by its motors. On the other hand, the Asguard robot reaches a COT half the Mini Cheetah one, due to its particular hybrid leg configuration.

The Mini Cheetah robot efficiency was evaluated for different values of the FhN neuron parameters m_0 and m_2 in Eq. 3.8 for which different leg motions are executed, impacting the COT value. A simulation campaign involving the Mini Cheetah was carried out to find the best CPG configuration to minimize the COT obtaining the best m parameters (m_0 and m_2) in Eq. 3.8) that can be selected based on the different terrains used as a testbed. Using the knowledge about the best CPG parameters, the results found are useful

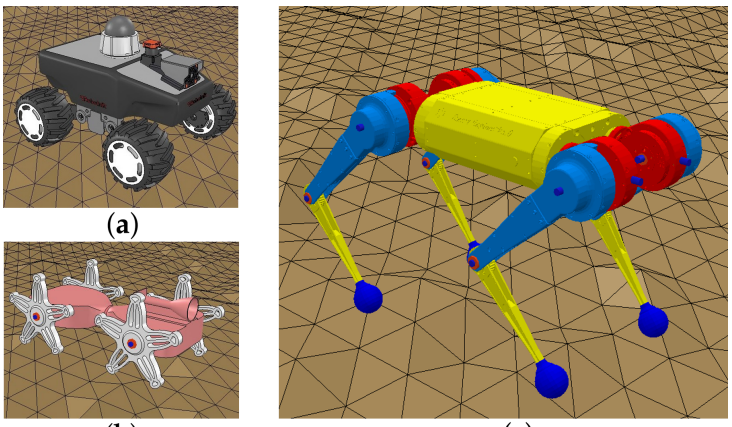


Figure 5.3: Different robotic structures taken into consideration for the tests in the complex terrain: at the top left the Robotnik, on the bottom left the Asguard, and on the right the Mini Cheetah.

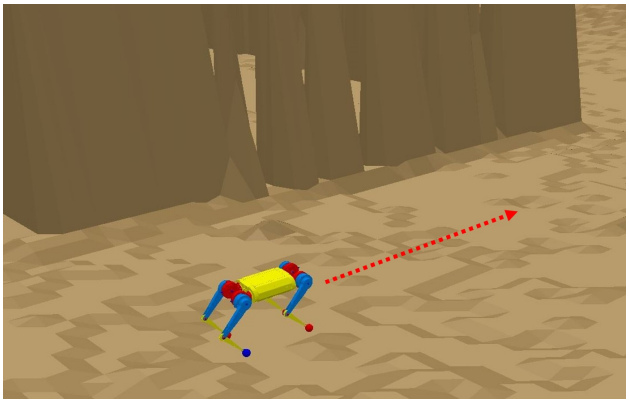
Table 5.2: COT index values related to the Asguard and Robotnik obtained on the uneven terrain reported in Fig. 5.2

Robots	**COT**		
	Flat	Uphill 15°	Downhill −15°
Asguard	0.57	0.71	0.63
Robotnik	1.26	1.47	1.79

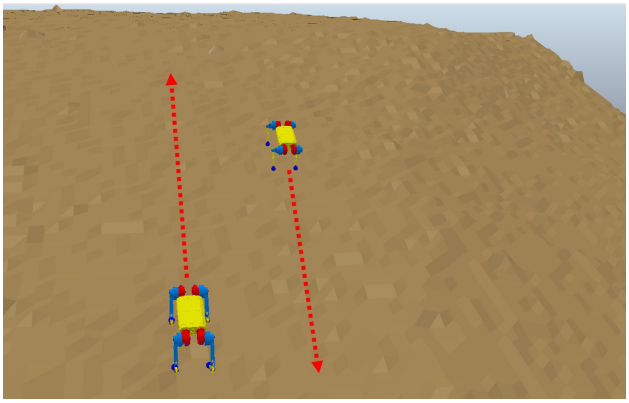
to design an adaptive control strategy that guarantees the lowest energy consumption for the robot during its path. The leg motions are therefore modulated in front of different terrain types, improving the COT. The terrain types considered are flat ground, uphill, and downhill.

To find an optimal value for the COT, a statistical approach is used. The COT distribution maps shown in Fig.5.5 can be obtained by changing, for each type of terrain, the m parameters and recording the COT values of five simulations each with a different robotic starting position. The yellow areas indicate parameter configurations that do not guarantee the stability of the robot locomotion whereas the yellow circle indicates the minimum value of COT.

Table 5.3 shows the optimal values obtained for flat ground, uphill (5° and 15°), and downhill (−5° and −15°) terrain. These COT results can be used as input for an adaptive control strategy that can modify the m parameters based on the current terrain type that can be estimated based on inertial sensors.



(a)



(b)

Figure 5.4: Different parts extracted from the terrain in Fig.5.2 used in the simulations: (a) flat; (b) uphill and downhill with a slope of ±15°.

Table 5.3: COT values and m parameters for the three different terrains: flat ground, uphill (5° and 15°), and downhill (−5° and −15°).

Terrain	m_0	m_2	**COT**
Flat ground	-2.5	-4.5	1.29
Uphill 5°	-3.5	-3	1.48
Uphill 15°	-4.5	-4	1.47
Downhill −5°	-4	-3.5	1.21
Downhill −15°	-2	-2	0.96

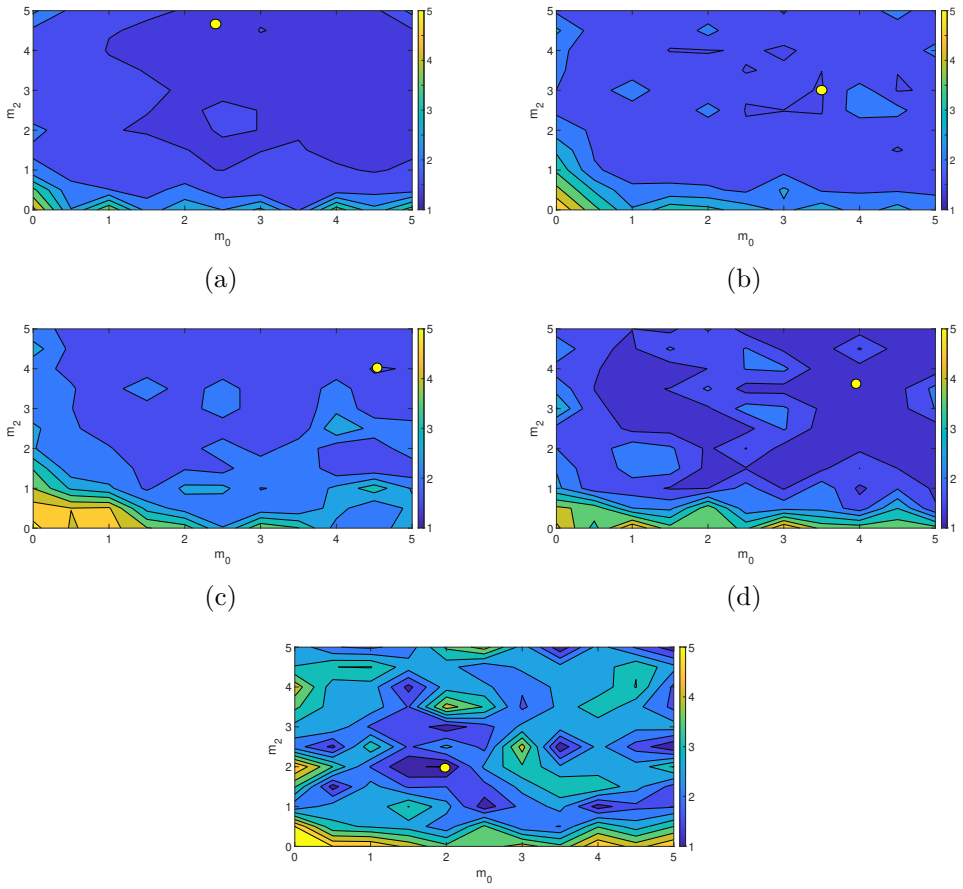


Figure 5.5: COT distribution maps depending on the combination of the parameters m_0 and m_2. (a) flat ground; (b) uphill 5° (c) uphill 15° (d) downhill −5° (e) downhill −15° . The yellow circle in the maps indicates the optimal value obtained.

5.2 Choice of the optimal neural parameters

The adaptive mechanism implemented in the Mini Cheetah is represented in the state machine in Fig.5.6. Developing tests in the simulation environment (Fig.5.2(b)) four pitch thresholds are defined: P_{UF}, P_{FU}, P_{DF}, P_{FD}. The pitch values of the robot body are called P, in particular:

- when $P < P_{UF}$ ($P > P_{FU}$) the robot passes from uphill (flat terrain) to flat terrain (uphill).
- when $P > P_{DF}$ ($P < P_{FD}$) the robot passes from downhill (flat terrain) to flat

terrain (downhill).

This strategy allows avoiding continuous passages between the three terrain classes, due to the noise present in the pitch signal unavoidable in realistic environments, creating a hysteretic behavior that guarantees a sharp class transition. A coupling parameter m is associated with each angular interval of the pitch, a suitable indicator for walking on the different slopes. The choice of parameters was made as a consequence of the results obtained in the previous simulations. Considering that for both uphills and downhills there are two possible m parameters, the pitch angle is analysed to see if it is closer to 5° than to 15° (or vice-versa) and consequently the m parameters are assigned.

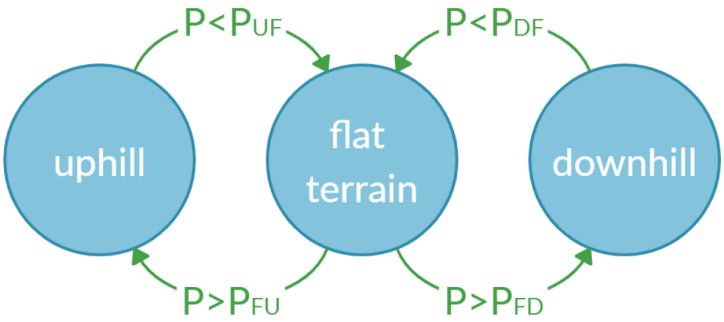


Figure 5.6: State machine used for the selection of the terrain class based on the robot pitch angles: an hysteretic function has been realised to avoiding continuous flickering between states ($P_{UF} = 0.08$, $P_{DF} = -0.10$, $P_{FU} = 0.16$, $P_{FD} = -0.14$).

5.3 Tests on a complex terrain

To test the COT performance of the Mini Cheetah, the terrain in Fig.5.7 is used. The path followed by the Mini Cheetah is composed of the previously reported fundamental sub-parts: flat ground, uphill 5°, uphill 15°, downhill −5° and downhill −15° that can be recognized by the quadruped using an inertial sensor. To compare the effects of the adaptive nullcline control on the COT result, a comparison is performed, considering also the case of the non-modulated nullcline slopes. Being the trot gait the most stable for quadruped locomotion at different speed profiles, it has been adopted for the tests. Locomotion speed can be controlled at the level of the neural oscillation, acting both on the neural oscillation frequency and, as in our novel approach, modulating the PWL slope values. However, the speed value is a consequence of the parameter modulation strategy focussing on minimizing the COT value. In fact, on uneven terrains, speed values are duly avoided by the optimization phase in that they can cause the robot to fall.

The relationship between power and velocity during the simulation reported in Fig.5.7 is shown in Fig.5.8. In particular, Fig.5.8(a) reports the data related to the fixed control mechanism, whereas Fig.5.8(b) shows the effect of the adaptive control strategy.

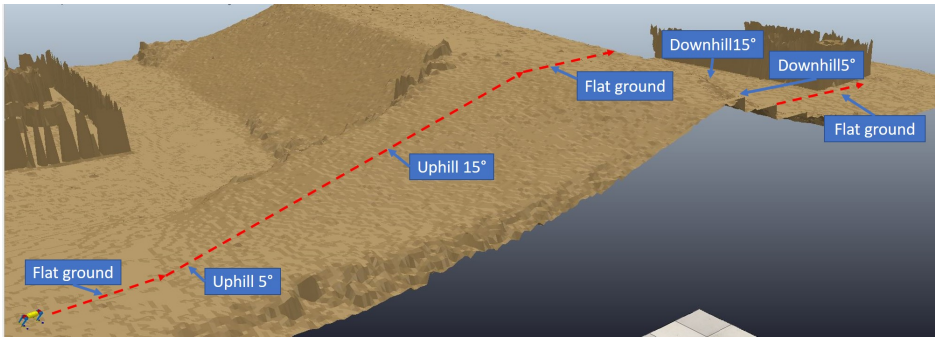


Figure 5.7: Terrain used as testbed in which the robot faces uphills, flat grounds, and downhills.

From the downhill $-15°$ to the uphill $15°$ case it can be seen that the power increases while the speed decreases, as expected, while flat locomotion lies in between. Each colored spot in Fig. 5.8 indicates the Power-vs-Speed values obtained from a $2s$ simulation of the robot in the corresponding terrain segment (flat, uphill, downhill). The substantial difference between the two cases regards the power recorded; in the case of fixed control, the samples cover the right part of the graph, the high-power zone, compared to the adaptive case. The analysis conclusion is that, in dealing with uneven and complex terrains, the adaptive control strategy decreases the overall amount of power required for all the cases analyzed. Fig.5.8(c) shows the comparison between the COT histogram for the adaptive and fixed cases while Fig.5.8(d) shows the COT statistical comparison for the adaptive and not adaptive cases. The statistical significance of the results obtained using the t-test is $p = 3.66 \cdot 10^{-12}$. Even if the COT standard deviation in the not adaptive case is more contained, nevertheless the average value (red line) is higher than in the adaptive case. This is also clear from the histogram representation in Fig.5.8(c): here the highest number of occurrences in the adaptive case is concentrated well below that one for the not adaptive one. Moreover, the larger distribution for the adaptive case can, in some cases, find COT values much lower than the not adaptive one. The improvement of the proposed strategy is significantly high concerning the fixed control method.

5.4 Conclusion

This chapter dealt with the energy consumption that a given quadrupedal, lightweight robot has to face to cross a complex path. For this reason, a detailed analysis to find the best configuration of the FitzHug-Nagumo's neuron control strategy aimed at minimizing energy consumption in the case of the Mini Cheetah robot was carried out.

Once having found the optimal m parameters governing the left and right FHN phase plane slopes, based on the particular terrain that the robot has to face, they can be used in a locomotion control strategy to change the robot gait in runtime. To develop this task an experimental analysis was carried out in simulation with the creation of a series of

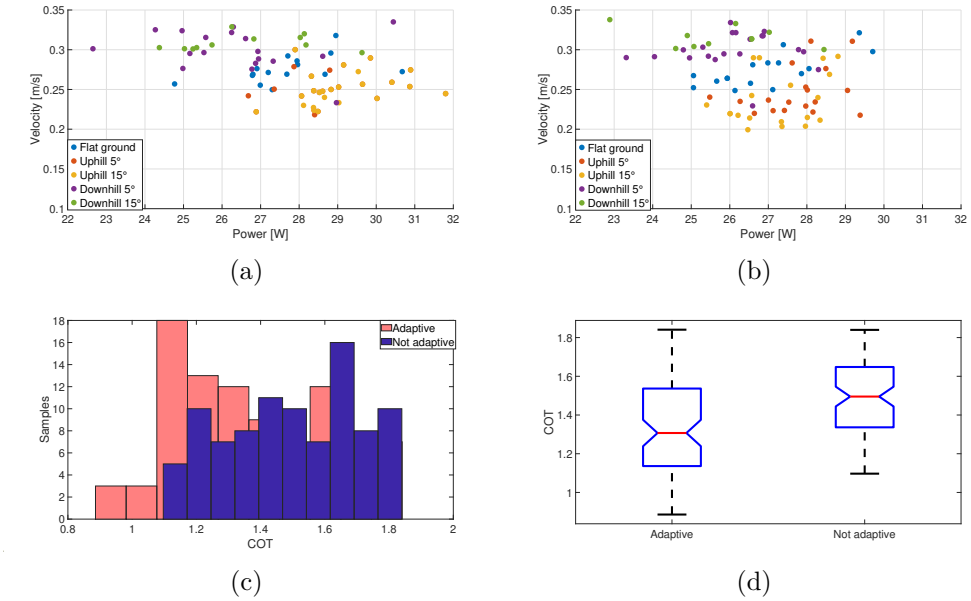


Figure 5.8: Point clouds describing the relation between velocity and power for the scenario reported in Fig.5.7 in the fixed (a) and adaptive (b) case; (c) histograms for the COT distribution; (d) statistical analysis of the COT index for the two analysed control strategies. On each box, the central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The whiskers extend to the most extreme data points not considered outliers, the notch indicates the 95% confidence interval of the median.

COT distribution maps and, consequentially, it was possible to create an adaptive control mechanism aimed at minimizing energy consumption.

Finally, a test terrain was provided to compare the adaptive control mechanism with the fixed control one demonstrating the improved performances that open the way to a reliable application of the quadruped robot in realistic scenarios reducing the constraints related to the battery life.

Chapter 6

MPC-based control strategies for the gait of a neuro-inspired quadruped robot[1]

In this Chapter, examples of the application of Model Predictive Control (MPC) based control strategies applied to a simulated neuro-inspired quadruped robot gait are reported. Differences in terms of performance will be underlined considering linear and nonlinear implementations. In particular, the advantages of a NNMPC approach will be underlined considering continuously time-varying trajectories and low friction surfaces along which the quadruped has to move. Part of the results of the chapter are reported in [31, 30].

6.1 MPC theory fundamentals

The MPC is a feedback control algorithm that uses a model to make predictions about future outputs of a process and it can handle Multi-Input Multi-Output (MIMO) systems. In other words, the goal of MPC is to optimize, over the manipulatable inputs, forecasts of a process trend. In order to realise this, forecasting is accomplished with a process model. But, in real applications, models are not perfect forecasters, and feedback can overcome some effects of poor models. The models that are controlled by MPC can be linear or nonlinear.

[1]The results reported in this chapter are extracted from "A data-driven neural network model predictive steering controller for a bio-inspired quadruped robot - Paolo Arena, Luca Patanè, Pierfrancesco Sueri and Salvatore Taffara - IFAC-PapersOnLine" and "MPC-based control strategy of a neuro-inspired quadruped robot - Paolo Arena, Pierfrancesco Sueri, Salvatore Taffara and Luca Patanè - 2021 International Joint Conference on Neural Networks (IJCNN)" and "Quadruped robot steering control on slippery surfaces via NNMPC - Paolo Arena, Luca Patanè and Salvatore Taffara - To be submitted".

A simpler way to control a system can be done using a PID controller, but, making a control action with n PID controllers can be challenging because each control loop would operate independently from the others as if there are no interactions between the n loops. Moreover, this strategy requires tuning too many controller gains. One of the advantages of MPC is that it is a multivariable controller that controls the outputs simultaneously by considering all the interactions between system variables. Besides, it is capable to handle constraints that are imposed on the input and output.

During the execution, only the first control action is used, the rest is discarded. In this way, the control action is based on the most recent value of the state of the plant. This strategy is called *state feedback strategy* and it makes the control system robust against model errors or changing of the reference signal [70, 71, 72].

In Fig.6.1, the relation between a static optimizer (representing the steady-state optimization), the MPC (dynamic optimizer), and the regulator (representing the low-level controllers) is shown. A static optimizer is used when the goal is finding those points (if any) at which a real-valued function has a minimum or a maximum [73], while a regulator is required when the plant can be controlled using a simple feedback control system. The MPC uses the set-points from the static optimizer while the measurements from the low-level controllers to which gives in input the actuator set-points.

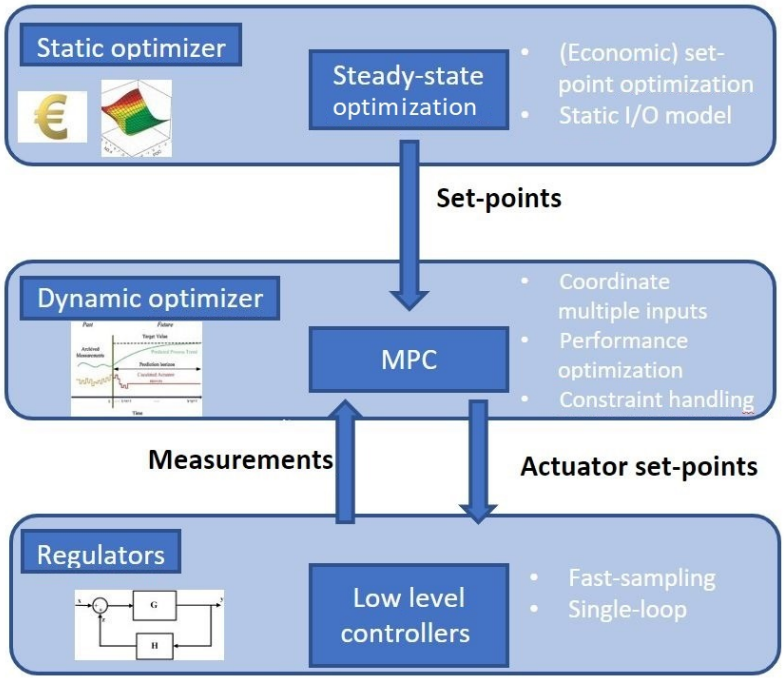


Figure 6.1: Relation between a static optimizer, the MPC, and a regulator-based control.

MPC will calculate the best controls within a trade-off of tracking the reference signal and a good control action. At each sample time, all the control actions inside a specific

interval are calculated. The limit of this interval is the so-called prediction horizon (N_2) which represents the number of time steps in the future for which the plant response is recursively predicted. Also, we will consider the variation of control action during a part of this interval, called control horizon (N_u) (see Fig.6.2). There is an important relationship between these two horizons: the control one must be under (or equal to) the prediction horizon ($N_u \leq N_2$).

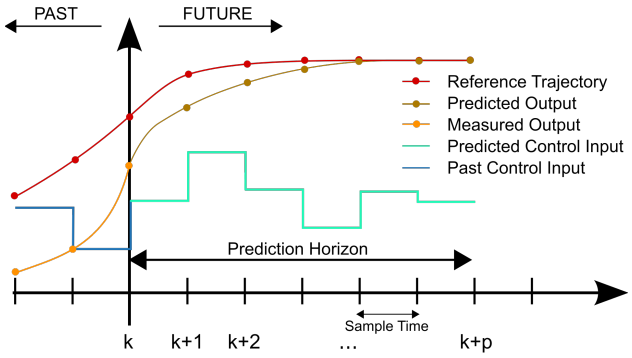


Figure 6.2: General control scheme of MPC.

Linear models in the process industries are, by their nature, empirical models and identified from input/output data. Considering a linear model in state-space form, an MPC can be represented in the state-space form:

$$\frac{dx}{dt} = Ax + Bu, x_{j+1} = Ax_j + Bu_j \tag{6.1a}$$

$$y = Cx, y_j = Cx_j \tag{6.1b}$$

where x is the n-vector of states, y is the p-vector of (measurable) outputs, u is the m-vector of (manipulatable) inputs, t is continuous time and j is the discrete-time sample number. The MPC in state-space form has several advantages, in particular, the easy generalization to multi-variable systems, the ease of analysis of closed-loop properties, and the online computation. Furthermore, considering the linear system theory, the linear quadratic regulator theory, Kalman filtering theory, internal model principle, etc., are immediately accessible for use in MPC starting with this model form. Categories, frameworks, and viewpoints, while indispensable for clear thinking and communication, may blind us to other possibilities. From a theoretical perspective, the significant shift in problem formulation came from the MPC practitioners who insisted on maintaining constraints, particularly input constraints in the problem formulation.

$$Du \leq d, Du_j \leq d \tag{6.2a}$$

$$Hx \leq h, Hx_j \leq h \tag{6.2b}$$

where D, H are the constraint matrices and d, h are positive vectors. The constraint region boundaries are straight lines as shown in Fig.6.3. It is assumed that $x = 0$, $u = 0$ is the steady state to which we are controlling the process.

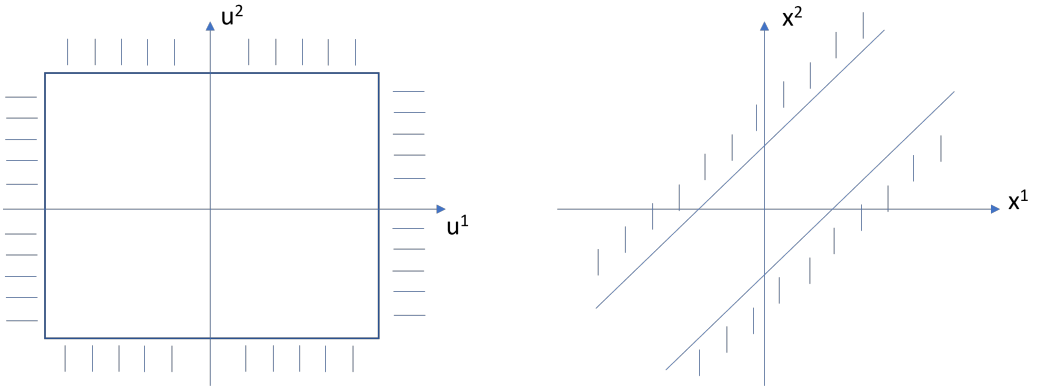


Figure 6.3: Example input and state constraint regions define by Eq.6.2(c-d).

Optimization over inputs subject to hard constraints leads immediately to nonlinear control, and that departure from the well-understood and well-tested linear control theory provided practitioners with an important and new control technology and motivated researchers to understand better this new framework.

6.1.1 Nonlinear MPC

For nonlinear plants, the ability of the MPC to make accurate predictions can be enhanced if a neural network is used to learn the dynamics of the plant instead of standard nonlinear modeling techniques. The selection of the minimization algorithm affects the computational efficiency of the algorithm.

The general scheme of the Neural Generalized Model Predictive Control (NGMPC) system can be seen in Fig.6.4. It starts with the input signal, $r(n)$, which is presented to the reference model. This model produces a tracking reference signal, $y_m(n)$, that is used as an input to the Cost Function Minimization (CFM) block. The CFM algorithm produces an output which is either used as an input to the plant or the plant's model. This input is set to the plant when the CFM algorithm has solved for the best input, $u(n)$, that will minimize a specified cost function. In the meanwhile, this input is set to the plant's model where the CFM algorithm uses this model to calculate the next control input, $u(n+1)$, from predictions of the response from the plant's model. Once the cost function is minimized, the input is passed to the plant.

Schematizing, the NGMPC algorithm follows these steps:

1. Generation of a reference trajectory. If the future trajectory of $y_m(n)$ is unknown, $y_m(n)$ is kept constant for the future trajectory.

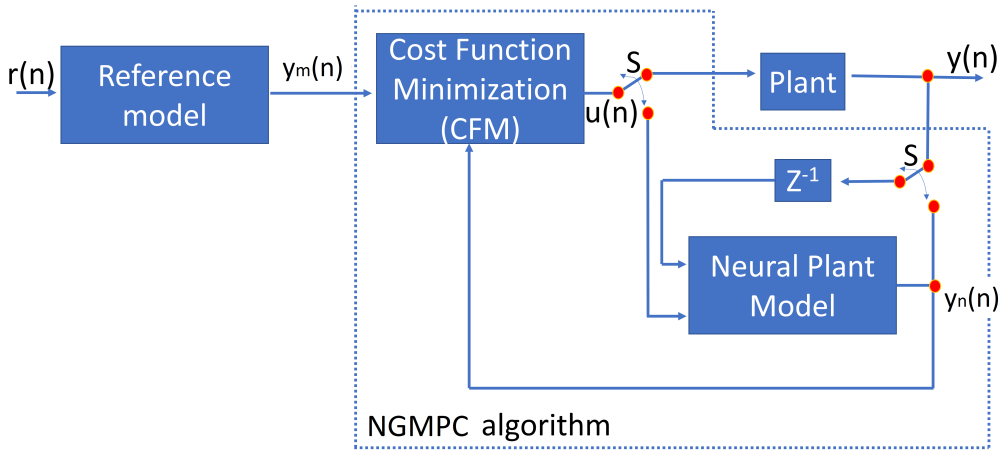


Figure 6.4: General scheme of a Neural Generalized Model Predictive Controller.

2. Starting with the previously calculated control input vector and predicting the performance of the plant using the model.
3. Calculation of a new control input that minimizes the cost function.
4. Repeat steps 2 and 3 until the desired minimization is achieved.
5. Send the first control input to the plant.
6. Repeat the entire process for each time step.

   The computational performance of a NGMPC implementation is largely based on the minimization algorithm chosen for the CFM block. The objective is to find a control time series that minimizes the cost function:

$$J_{NGMPC} = \sum_{j=N_1}^{N_2} [y_m(n+j) - y_n(n+j)]^2 + \sum_{j=1}^{N_u} \lambda(j)[\Delta u(n+j)]^2 \qquad (6.3)$$

where N_1 is the minimum cost horizon. N_2 is the maximum cost horizon, N_u is the control horizon, y_m is a reference trajectory, y_n is the predicted output of the neural network, λ is the control input weighting factor and $\delta u(n+j)$ is the change in u and it is defined as $u(n+j) - u(n+j-1)$.

This cost function minimizes the mean squared error between the reference signal and the plant's model, and also the weighted squared rate of change of the control input. When this cost function is minimized, a control input that meets the constraints is generated that allows the plant to track the reference trajectory within some tolerance.

There are four tuning parameters in the cost function, N_1, N_2, N_u, and λ. In particular, the predictions of the plant will run from N_1 to N_2 future time steps, while the bound on the control horizon is N_u. The second summation contains a weighting factor λ that is introduced to control the balance between the first two summations. The weighting factor acts as a damper on the predicted $u(n+1)$.

Nonlinear models are used in MPC to improve the quality of the forecasting. The fundamentals in any process control problem (conservation of mass, momentum, and energy, considerations of phase equilibria, relationships of chemical kinetics, and properties of final products) introduce nonlinearity into the process description. For processes operated over large regions of the state space (semi-batch reactors, frequent product grade changes, processes subject to large disturbances) the advantages of nonlinear models appear larger.

Regardless of the identification method, the nonlinear model inside the MPC controller also in state space form is given by Eq. 6.4.

$$\frac{dx}{dt} = f(x,u), x_{j+1} = f(x_j, u_j) \tag{6.4a}$$

$$y = g(x), y_j = g(x_j) \tag{6.4b}$$

$$u \in U, u_j \in U \tag{6.4c}$$

$$x \in X, x_j \in X \tag{6.4d}$$

If the model is nonlinear, there is no advantage in keeping the constraints as linear inequalities, so the constraints are taken as membership in more general regions U, X shown in Fig.6.5.

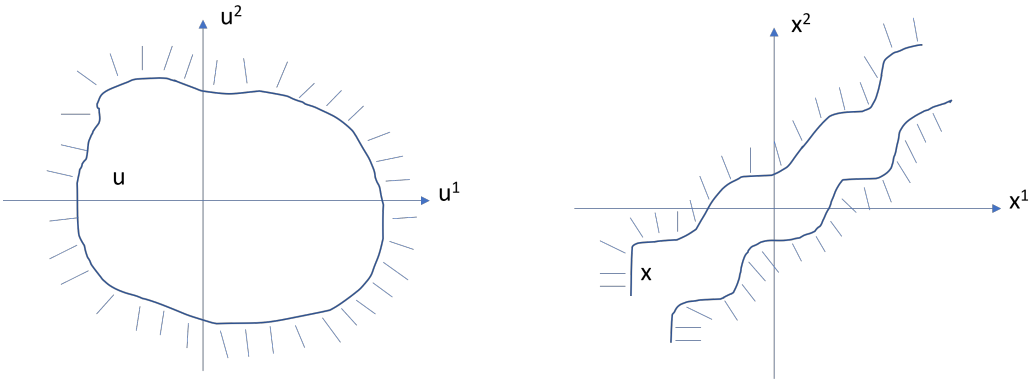


Figure 6.5: Example input and state constraint regions defined by Eq.6.5

The quality of the plant's model affects the accuracy of a prediction. A reasonable model of the plant is required to implement GMPC. With a linear plant, there are tools and techniques available to make modelling easier, but when the plant is nonlinear this task is more difficult. Currently, there are two techniques used to model nonlinear plants. One is to linearize the plant into a set of operating points. If the plant is highly nonlinear the set

of operating points can be very large. The second technique involves developing a nonlinear model which depends on making assumptions about the dynamics of the nonlinear plant. If these assumptions are incorrect the accuracy of the model will be reduced. Models using neural networks have been shown to have the capability to capture nonlinear dynamics. For nonlinear plants, the ability of the GPC to make accurate predictions can be enhanced if a neural network is used to learn the dynamics of the plant instead of standard modelling techniques. Improved predictions affect rise time, overshoot, and the energy content of the control signal [74].

Optimization problem and solver

To determine inputs of a given process that optimises the forecasted process behaviour, these inputs, or control actions, are calculated repeatedly using a mathematical process model for the prediction. In doing so, the fast and reliable solution of convex quadratic programming problems in real-time becomes a crucial ingredient of most algorithms for both linear and nonlinear MPC. So, to obtain the optimal control action, an optimization problem has to be constructed and be solved; for this purpose, it is needed a constrained quadratic programming problem, in particular a Quadratic Programming (QP) problem. This kind of problem has the form:

$$\min_{\mathrm{x}} \frac{1}{2} x^T H x + x^T g(w_o) \tag{6.5a}$$

$$s.t. \quad lbA(w_o) \leq Ax \leq ubA(w_o) \tag{6.5b}$$

$$lb(w_o) \leq x \leq ub(w_o) \tag{6.5c}$$

where H is a positive (semi-)definite Hessian matrix of the objective function, g is the gradient of the objective function, A is the constrain matrix, lbA and ubA represent the lower and upper bounds for the constrain matrix respectively, lb and ub represent the lower and upper bound for the optimization variable, respectively.

6.2 MPC-based quadruped locomotion

As introduced in Section 3.3, MPC strategies are used to implement an optimal control for environmentally induced steering control. In particular, MPC is requested to generate the descending commands to the low level CPG controller [30, 31]. In order to compare the performance given by the linear and nonlinear MPC approaches, different simulations are carried out to develop a comparative analysis. The differences between the two methods will be highlighted. Finally, the obtained results are analyzed and compared with those obtained in the same robotic architecture using a simple standard PID controller.

6.2.1 LMPC and NNMPC design for a quadruped robot

The robot architecture and neural control algorithm, presented in Section 3.1 will be used in this section. An overview of the results obtained in terms of gait stability when a linear or a nonlinear MPC approach is used will be provided in the following part. The MPC parameters used are summarized in Table 6.1. In the table, the characteristic parameters of MPC are the prediction and control horizon. The former, according to the MPC guidelines, was chosen to have 20-30 samples covering the open-loop transient system response, whereas the latter has the best trade-off choice from 10% to 30% of the Prediction Horizon.

Table 6.1: MPC parameters

MPC parameters	**Values**
Sample time	0.05s
Number of manipulated variables	1 (steering)
Number of the measured outputs	1 (yaw)
Prediction horizon	150
Control horizon	40
Closed-loop performance	fixed
Speed of state estimation	fixed

LMPC approach

In order to implement a LMPC, the development of a linear model of the process to be controlled is required. The quadruped robot model is identified with a data-driven approach obtained using the Matlab identification toolbox. In particular, the relation between the input that corresponds to the steering command provided to the robot, in the form of the $feed3_{fi}$ signals (Eq. 3.14), and the output that corresponds to the robot's yaw angle, is modelled through a transfer function reported in the following

$$Fdt = \frac{(3.54)s^4 + (6.04)s^3 - (13.71)s^2 - (0.008)s - (0.005)}{s^5 + 161.26s^4 + 212.81s^3 + 1.46s^2 + 0.08s + 1.02 \cdot 10^{-4}} \tag{6.6}$$

To identify the robot model, five different datasets have been used, one to identify the model and the others to test it. The sampling time is fixed to 50 ms. The relations between the steering and yaw values are shown in Fig.6.6. The training dataset is shown in Fig.6.6(a) and it is used for the identification. Its robot behaviour is characterized by a yaw ramp with a negative slope, followed by a ramp with a positive slope, and finally a more complex behaviour with rapid changes of heading. A total of 10000 samples is considered. The steering control action S_c in Eq.3.14 covers the entire allowed range (i.e.

from -1.2 to 1.5) which is a hard constraint used for the MPC output signal. The first and the second test datasets are reported in Fig.6.6(b) and Fig.6.6(c). Other three datasets where the robot is following square-like trajectories were considered during the test phase. To maintain a straight trajectory a steering signal (i.e. S_c) of about 0.2 is needed. The linear model has been developed considering eleven different structures for the transfer function with different numbers of poles and zeros.

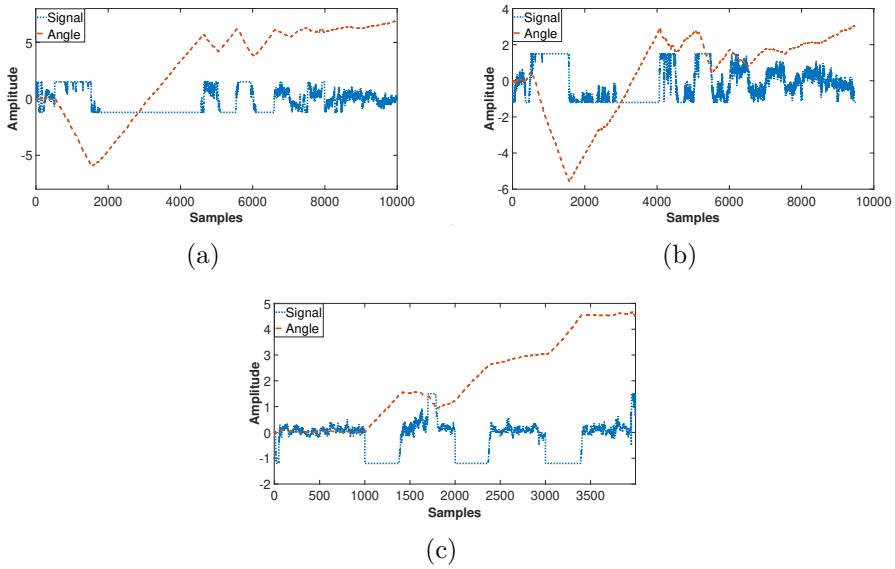


Figure 6.6: Training and test datasets used for the linear model identification considering as input the S_c signal and as output the yaw angle. (a) Training dataset; (b-c) Test datasets. The red line represents the Yaw Output, the blue line is the steering control action. The yaw angle is reported in radians.

In Fig.6.7, an high-level scheme of the system architecture is reported. The MPC inputs are the reference signal (i.e. x_{ref}) and the actual yaw angle acquired from the robot (i.e. *mo* in Fig 6.7). The control input provided to the robot in terms of steering is the output of the MPC controller. The communication with the CoppeliaSim framework has been performed using the 2-level s-function available in Matlab.

NNMPC approach

As for the linear case, to identify the robot model, five different datasets have been used, one to identify the model and the others to test it and the data-driven approach is used to identify the robot model.

Similarly to the LMPC case, in Fig.6.8 a high-level scheme of the system architecture, developed in Matlab-Simulink, is reported. The reference signal X_{ref}, representing the desired yaw, is an input for the NN Predictive Control block, together with the actual

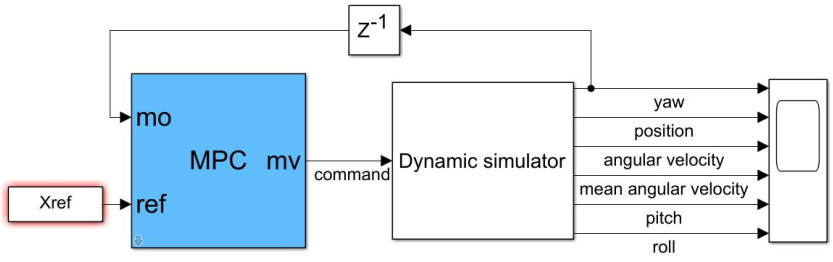


Figure 6.7: Scheme of the LMPC developed in Simulink.

yaw of the robot, whereas the output is the control signal S_c which modulates the CPG locomotion network. The communication with the CoppeliaSim dynamic framework has been performed using the 2-level s-function available in Matlab. The MPC parameters used are the same of those used in the LMPC case.

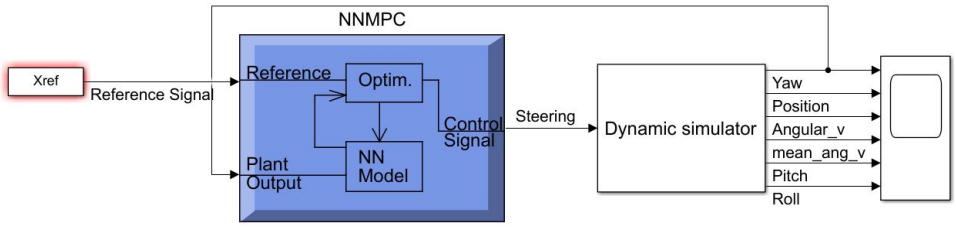


Figure 6.8: Scheme of the NNMPC developed in Simulink.

The NNMPC block employs a Neural Network-based model of the robot behaviour able to describe and predict the effect of the steering control signal on the robot heading. The best control sequence is derived via a numerical optimization algorithm minimizing the following performance criterion J over the specified horizon:

$$J = \sum_{j=1}^{N}(y_r(t+j) - y_m(t+j))^2 + \rho \sum_{j=1}^{N_U}(u'(t+j-1) - u'(t+j-2))^2 \qquad (6.7)$$

where N defines the cost horizons over which the tracking error is evaluated and N_U defines the dimension of the control horizon, in which the control increments are analyzed. The u' variable is the tentative control signal, y_r is the reference (desired response), while y_m is the neural network model response. The ρ value weights the contribution of the sum of the squared control increments on the index J; its value was fixed to 0.05. The

optimization block determines the control input u' minimizing J, and then the optimal u is provided as input to the robot CPG controller. The minimization algorithm used is *srchbac* and it is a one-dimensional minimization routine based on a backtracking technique [75].

The Search Parameter α works as a stop criterion for the minimization routine: if the minimization between two consecutive control input candidates is less than α, the routine stops.

As said, the linear model used to set the MPC was obtained by analyzing different transfer function structures, containing a varying number of poles and zeros (from 1 to 7). The outcome of this identification procedure was a transfer function characterized by 5 poles and 4 zeros. Based on this analysis, for the design of the nonlinear model, the regressors for the input and output variables were used as input features for the neural network structure. Therefore the input layer of the networks consisted of a nine-dimension vector. The network has been trained offline in batch mode, using the training dataset previously defined and adopting the Levenberg-Marquardt learning technique with a maximum number of epochs equal to 400. To find the best number of hidden neurons a supplementary analysis was carried out, in which the number of neurons was varied in the range $[3, 12]$.

The results of this hyperparameter analysis are shown in Fig.6.9.

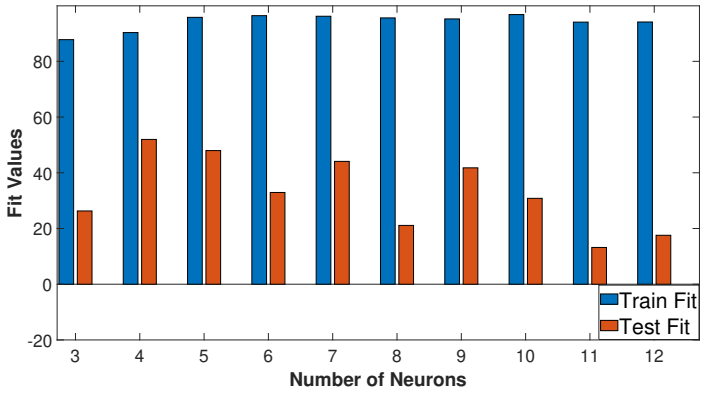
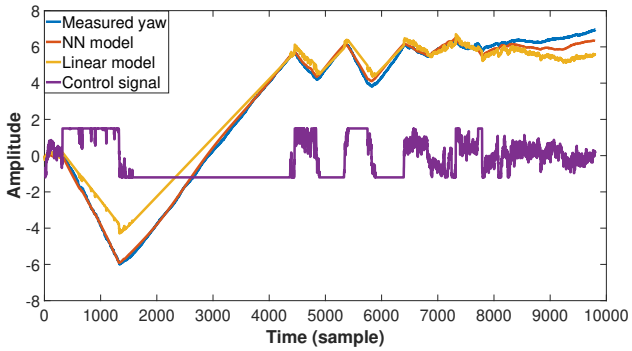
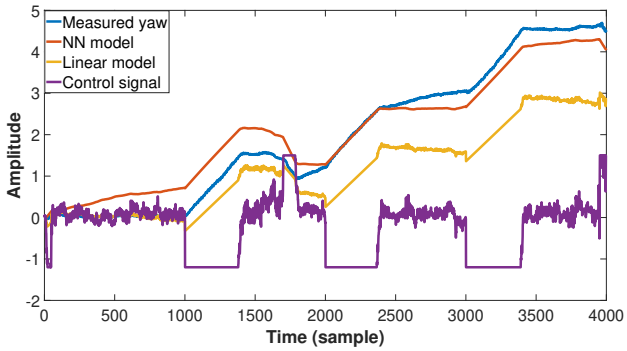


Figure 6.9: Network performance as a function of the number of hidden neurons. The blue bars represent the Fit_{NRMSE} on the training dataset, while the red bars represent the mean of the Fit_{NRMSE} calculated on the test datasets.

The most suitable number of hidden neurons selected is four. This network has a training fit value of 94 and a mean test fit value of 65, which represents an improvement if compared to the linear model identification, which was able to reach fit values in training and test of 78 and 52, respectively. In Fig.6.10 the comparison between the neural model and the linear transfer function is reported.



(a)



(b)

Figure 6.10: Comparison of the identification performance for a linear and nonlinear model on the (a) training dataset and (b) one of the test datasets.

Results and comparisons

The NNMPC and LMPC were tested on six different reference signals: two representing an ideal square route (one clockwise and the other counterclockwise), two representing an ideal equilateral triangle route (also in this case, in both directions), a one-period sine reference and two semicircular routes.

The results related to the linear MPC version, are reported in Fig.6.11. For the nonlinear MPC version, the results are reported in Fig.6.12, as a function of the searching parameter α, used as a stopping criterion. In particular, in Fig.6.12(c), when the first negative step is applied in the reference yaw, the controller does not properly work with $\alpha = 0.001$. Here, the control signal starts to oscillate between the two saturation limits with the result that the output yaw is almost constant and loses the reference command tracking.

To better evaluate these results, the Fit_{NRMSE} between the robot yaw and the reference signal was evaluated. The fit value is affected by the maximum angular velocity

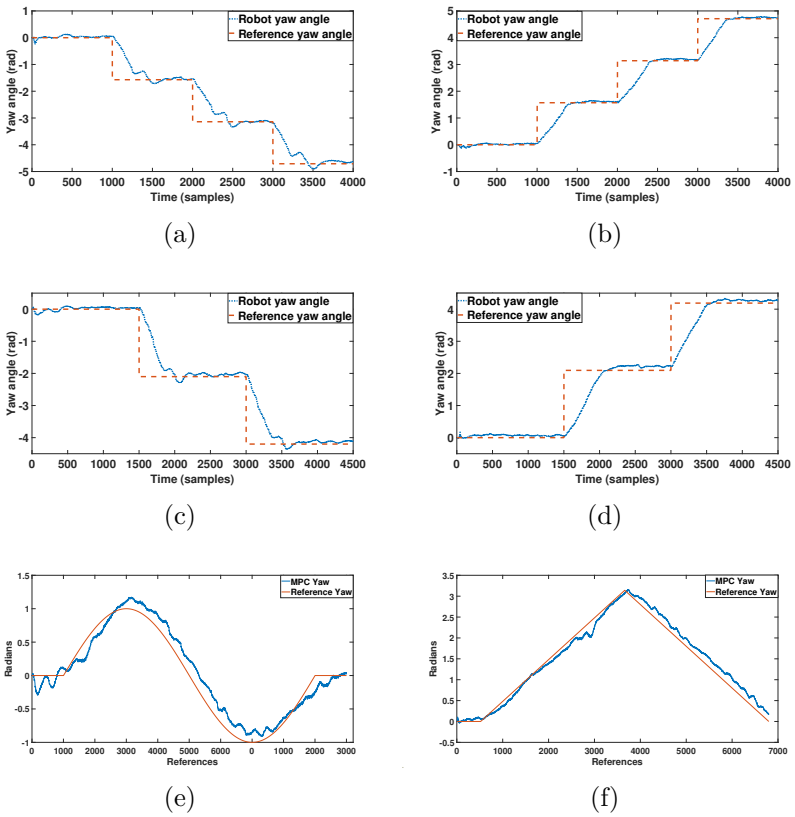


Figure 6.11: Behaviour of the robot controlled by the Linear MPC on: (a) Clockwise square reference; (b) Counterclockwise square reference; (c) Clockwise triangle reference; (d) Counterclockwise triangle reference; (e) Sine yaw reference; (f) Semicircular reference.

that the robot can reach, due to the physical constraints and to the constraints added in the control action, i.e. the steering bounds. The reference signal is an ideal figure, and it assumes that the robot should have the capability to turn on the spot: this is not possible in our case, unless at the expense of complicated maneuvers, not conceived within the trotting gait, which is the locomotion pattern shown by our robot.

Table 6.2 shows the outcome of this analysis: the best results were obtained with the search parameter α equal to 0.01 (i.e., Mean $Fit_{NRMSE} = 77$).

Finally, the NNMPC and the LMPC performance were compared. To test the MPC performance, a comparison with the PID controller is carried out. The control scheme is similar to the one shown in Fig.6.7 with the PID block instead of the MPC block. The results in terms of Fit values and maximum absolute error (MaxAE) are outlined in Table 6.3. The NNMPC performs better in almost all cases as also demonstrated in Fig.6.13. The accuracy is even more evident when the yaw signal undergoes continuous variations, as depicted in Fig.6.13 (e)-(f). Here the NMPC controller does not suffer from any tracking

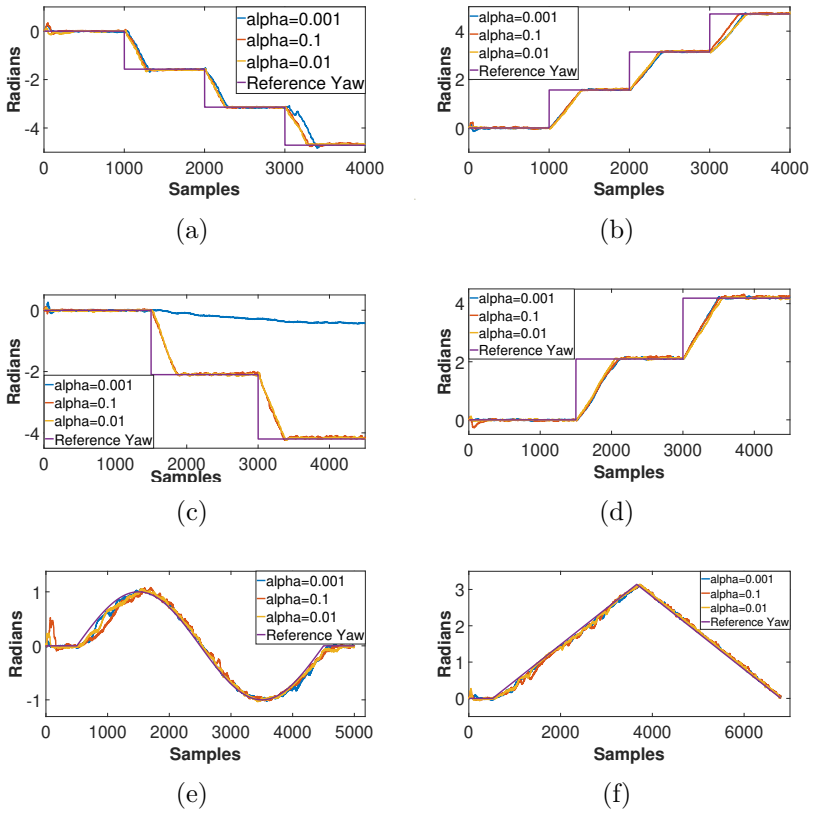


Figure 6.12: Behaviour of the quadruped robot while following the reference steering command generated by the NNMPC with α varying between 0.1, 0.01, 0.001: (a) Clockwise square reference; (b) Counterclockwise square reference; (c) Clockwise triangle reference; (d) Counterclockwise triangle reference; (e) Sine reference; (f) Semicircular reference.

delay and is also smoother than the MPC controller result.

Concluding, the comparative analysis carried out considering the fit values performance and the MaxAE index demonstrates the effectiveness of the NNMPC in particular in presence of continuously time-varying trajectories.

6.3 Locomotion in low friction surfaces

The comparison between LMPC and NNMPC carried out in the previous part of the chapter can be extended considering the interaction between the quadruped and the surface. The goal is to show the limits of the linear model when the robot is in presence of slipping surfaces. In this case, the use of a nonlinear model is essential to capture the intrinsic frequencies of the robot oscillations, due to the stepping locomotion. In the following

Table 6.2: Fit values for the six datasets obtained through the LMPC, for different values of α. The mean of the fit values is included in the last row.

References	Fit_{NRMSE} α=0.1	Fit_{NRMSE} α=0.01	Fit_{NRMSE} α=0.001
CW sq	74.51	75.77	70.35
CCW sq	71.60	69.57	69.67
CW tr	71.44	70.49	-44.38
CCW tr	64.48	64.26	64.16
sine	80.87	88.97	88.69
semi	90.04	92.85	92.57
Mean	**75.49**	**76.99**	**56.84**

Table 6.3: Comparison between the NNMPC and LMPC Fit values and MaxAE for the six test datasets adopted. In the last row, the mean values calculated on the datasets are indicated. (CW sq: clockwise square; CCW sq: counterclockwise square; CW tr: clockwise triangle; CCW tr: counterclockwise triangle; sine: sine yaw; semi: semicircular).

Ref	**NNMPC**		**LMPC**	
	Fit	MaxAE	Fit	MaxAE
CW sq	75.77	0.20	72.31	1.63
CCW sq	69.57	0.24	71.02	1.58
CW tr	70.49	0.19	68.48	2.18
CCW tr	64.26	0.28	65.45	2.06
sine	88.97	0.08	74.88	0.30
semi	92.85	0.07	84.47	0.40
Mean	**76.99**	**0.17**	**72.77**	**1.35**

simulations, the quadruped locomotion is governed using Eq. 3.3 introduced in chapter 3.

Using the NNMPC is possible to build a black-box model of the system where the nonlinear characteristics can be learned from simulation and/or experimental data. In this work, a neural model is shown to autonomously capture the nonlinear interactions between the ground reaction force in slippery conditions and the center of mass motion. The model can be therefore used within the MPC controller to generate an efficient control of the robot's motion.

The low-level CPG locomotion controller is here adopted to generate a set of reference phase relationships among the actuators of the robot legs. It is based on the Reaction-Diffusion Cellular Neural Network (RD-CNN) [76] paradigm, able to show steady-state stable phase-shifted dynamics to generate the desired locomotion gait [15, 77]. Here, an efficient steering control is realised by applying suitable gains to the neuron links of the

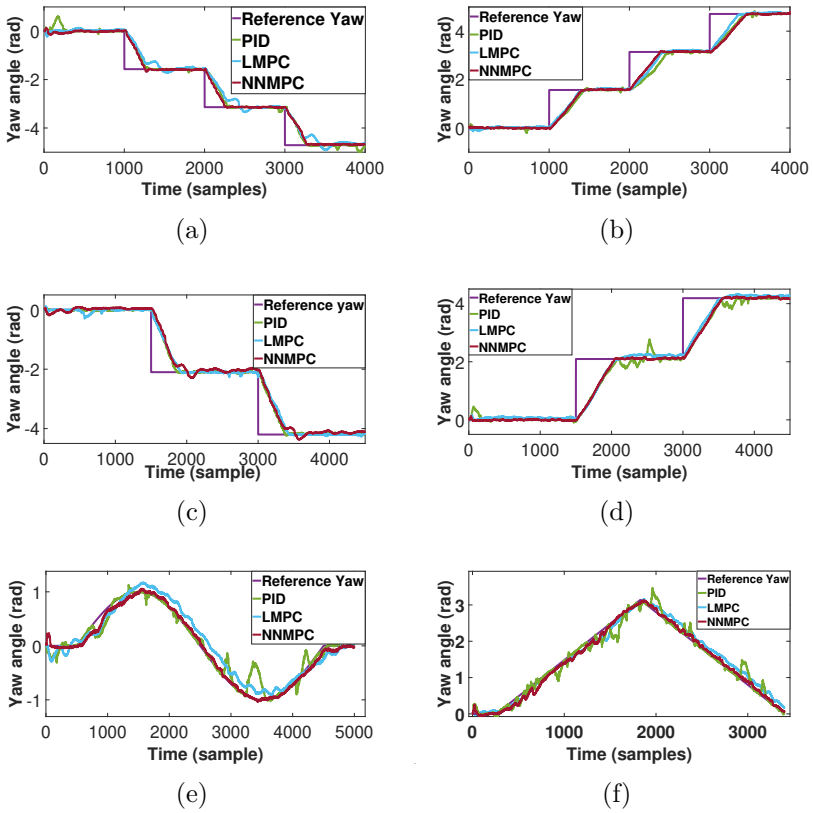


Figure 6.13: Comparison between the robot heading and the reference signal obtained using the LMPC, the PID, and the NNMPC control strategies in the case of: (a) Clockwise square reference; (b) Counterclockwise square reference; (c) Clockwise triangle reference; (d) Counterclockwise triangle reference; (e) Sine reference; (f) Semicircular reference.

RD-CNN. The phase stability of the steering gait is rigorously demonstrated by exploiting Partial Contraction Theory [78]. Thanks to these results, a unique gain K_L can be fixed to reach a stable locomotion pattern invariant with respect to the desired steering angle which can be imposed through suitable gains $w_{i,j}$ among the network links. The high-level controller will be in charge of the selection of the suitable w_{ij} for a proper steering.

6.3.1 MPC and NNMPC design

The standard strategy to design the MPC controller both in the linear and in the nonlinear case was followed. In Fig.6.14 a high-level scheme of the system architecture, developed in Matlab-Simulink, is reported in the case of the NNMPC implementation. The only difference with the LMPC is the block outlined as "NN predictive controller", which, in the linear case, employs a linear model. The reference signal *sig*, representing the desired

yaw speed of the robot center of mass ω_z, is an input for the NN Predictive Control block, together with its actual value. The controller output consists, in principle, of the vector $\mathbf{w}$ which, applied in Eq.3.3 realizes the low-level steering control in the trotting Mini Cheetah. The output weight vector is reduced to a scalar value applied only to specific leg joints. So the MPC output is only one gain value. Communication with the CoppeliaSim dynamic framework has been performed using a 2-level s-function available in Matlab.

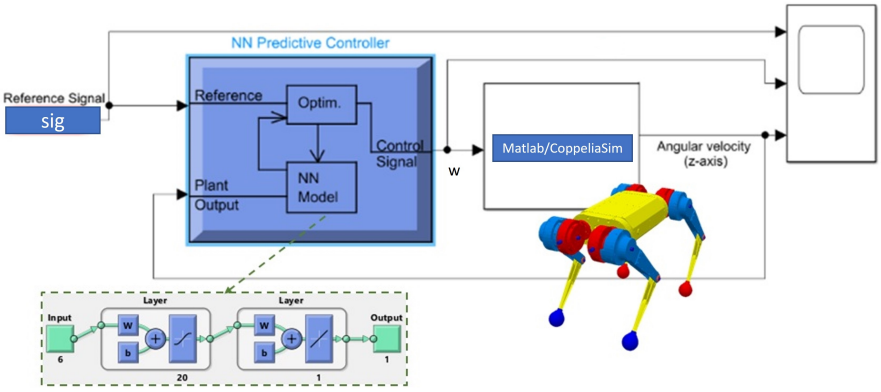


Figure 6.14: High-level scheme of the NNMPC-based architecture for steering control.

Within the NNPC block (Fig.6.14) the robot neural model receives as input the steering control signal (i.e. the gain $\mathbf{w}$) whereas the output is the heading (i.e. the yaw velocity, ω_z) of the robot center of mass. Once the model is obtained, the best control sequence is calculated by minimizing the following performance index J over the given horizon:

$$J = \sum_{j=1}^{N_H} (y_r(t+j) - y_m(t+j))^2 + \rho \sum_{j=1}^{N_U} (w'(t+j-1) - w'(t+j-2))^2$$

$$s.t. \ \ |\mathrm{w'(t)}| < \gamma \tag{6.8}$$

where according to the classical MPC formulation: N_H is the prediction horizon where the tracking error is evaluated; N_U is the control horizon where the control signal samples are analyzed; w' is the tentative control signal; y_r is the reference response; y_m is the model output; ρ (in this case $\rho = 0.05$) weights the contribution of the sum of the squared control increments on the index J; $\gamma = 0.07$ is the saturation limit of the control input. The optimization block determines the control input w'_0 minimizing J, to be provided to the robot CPG controller. The minimization algorithm used is *srchbac*, a one-dimensional minimization routine based on a backtracking technique [79]. According to the MPC

guidelines, N_H was chosen to have 20-30 samples covering the open-loop transient system response, while N_U is a fraction $(10\% - 30\%)$ of N_H. Typically, only the control input at the next time step is applied to the robot, to make the control law more effective.

The MPC parameters adopted are reported in Tab.6.4.

Table 6.4: Parameters used in the NN Predictive Controller block shown in Fig.6.14

Parameters	**Value**
Sample time	0.05s
No. manipulated variables	1 (w)
No. measured outputs	1 (ω_z)
Prediction horizon N_H	150 samples
Control horizon N_U	40 samples

Once the dataset was generated, the relation between steering command and angular velocity can be modelled using either a continuous-time transfer function (LMPC) or a neural network (NNMPC).

6.3.2 Performance in slippery conditions

A slippery condition takes place whenever a foot in the stance phase shows a non-zero velocity with respect to the world reference frame [80]. To quantify the efficiency of the controller with respect to slippery, in front of the different friction conditions, the following *Slipping Index* (SI) is defined as follows:

$$SI = \|P_{fs}^{W}(t) - P_{fs}^{W}(t-1)\| \tag{6.9}$$

where $P_{fs}^{W}(t)$ is the planar vector connecting the generic robot foot position, when in the stance phase, to the World reference frame W. The index takes into account the difference between two consecutive planar positions of a specific foot in contact with the ground. Considering the time between two position recordings as unitary, SI accounts for the residual speed of the robot foot. The idea behind this definition is that the generic foot, while in stance, should not move, in ideal conditions and in case of high friction. In this chapter, since a simulation approach is adopted, this index can be reliably evaluated with respect to the world reference frame W. In the case of a real robot, it can be easily referred to as the robot reference frame using well-known kinematic relations. In real conditions, but also when using a realistic dynamic simulation environment, the index never goes to zero, but maintains around a certain small quantity, recording small natural fluctuations. So, if SI maintains below a certain upper bound $\overline{SI}$, the slipping conditions can be neglected, otherwise slipping has to be considered a source of potential instability. This index, due to the dynamic conditions, has to be experimentally tuned both in simulations and in the real robot deployment.

The software tools employed to perform the quadruped robot simulations are Matlab, Simulink, and CoppeliaSim. In particular, the control scheme used to drive the robot is

implemented in Simulink. The dataset acquisition campaigns, the model generation for the NMPC and LMPC, and the locomotion gait generation are implemented in Matlab. In CoppeliaSim the legged robot and its interaction with the adopted terrains can be modelled, simulated, and controlled through Matlab.

Fig.6.15 shows the steering behaviour of the Mini Cheetah model within the simulated environment in which the tests are performed. In Table 6.5 the physical characteristics of the modelled robot are reported.

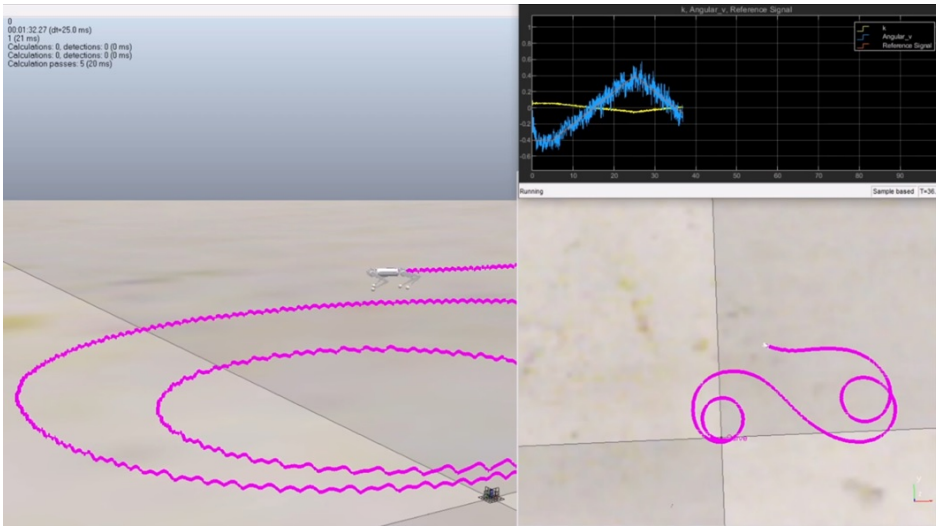


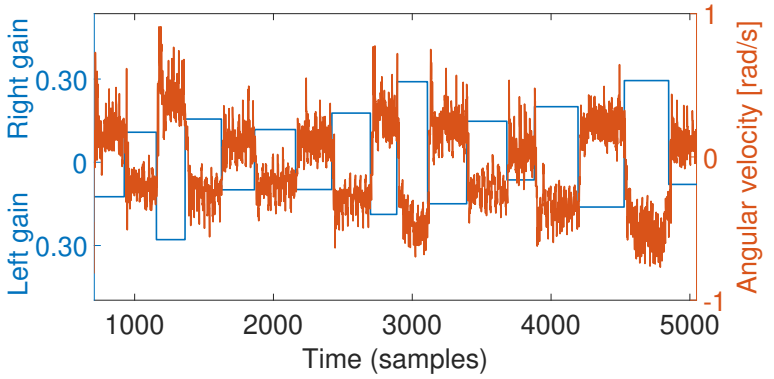
Figure 6.15: The simulated Mini Cheetah robot is shown while performing steering manoeuvres in the CoppeliaSim framework. A trace of the center of mass position is left by the robot during the simulation as shown in the lateral (left panel) and top view (bottom right panel) of the scene. The reference signal and the actual steering speed are also shown in the top right panel.

Table 6.5: Mini Cheetah robot physical characteristics adopted in the simulated model.

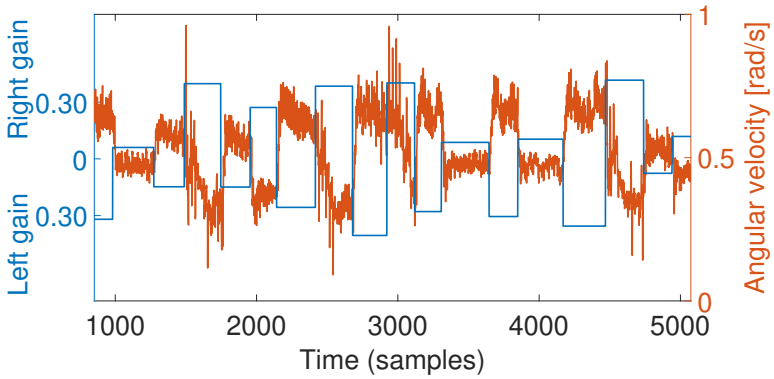
Features	**Values**
Height	30 cm
Length	48 cm
Width	27 cm
Weight	9 kg

The effect of the gain-based steering control in terms of steering velocity is reported in Fig.6.16(a) where a portion of the learning dataset (entirely composed of 500 steps with random amplitude and duration) is shown in normal friction conditions that corresponds to the interaction between rubber and dry asphalt.

A different behaviour appears in presence of low friction that can be considered equal to the interaction between rubber and wet ice, as shown in Fig.6.16(b). The complexity of the low friction model hidden within the data will be evident when linear and nonlinear modelling techniques for designing the MPC will be compared.



(a)



(b)

Figure 6.16: Dataset obtained in the dynamic simulation environment in which couples w-ω_z are collected from the walking quadruped robot: (a) normal friction; (b) low friction.

6.3.3 CPG and steering mechanism

The CNN-based CPG approach adopted for the generation of locomotion patterns for the considered quadruped robot, guarantees stable solutions when the constraints formulated using the partial contraction theory are satisfied. Referring to Eq. 3.7, the direct calculation of the maximum eigenvalue of the Jacobian matrix, evaluated over the limit cycle

shown by the single, uncoupled cell, leads to:

$$\lambda_{max}\left(\frac{\partial f}{\partial x}(x_i,t)\right) = 0.646$$

whereas the algebraic connectivity of the graph is: $\lambda_1 = 0.2679$. Therefore, we can choose k=3 to satisfy Eq. 3.7.

Considering the FL leg as the reference one, a specific gait is defined through the following vector:

$$\Theta_{gait} = [\theta_{FL,FL}; \theta_{FL,FR}; \theta_{FL,BL}; \theta_{FL,BR}]$$

establishing the phase shift of all the legs with respect to the reference one. In this case, the trot gait was adopted, characterised by:

$$\Theta_{trot} = [0°; 180°; 180°; 0°] \tag{6.10}$$

The robot steering, when following a path, is performed exploiting the w_{ij} entries of the vector **w** in Eq. 3.3. As stated above, only one gain value is calculated by the controller and applied to a subset of leg joints. In detail, in our case $W = \{w_{i,j}\} \in R^4$ where $i = \{1, 2\}$ and $j = \{FL, FR, BL, BR\}$, if the same scaling value $\bar{w}_{ij}$ is imposed to the state variables of ipsilateral (e.g. right) legs, (right) the steering is performed, maintaining the same gait (i.e. trot), given the phase invariance imposed by the Laplacian couplings. Also, in our case the maximum steering radius is obtained via $w_{1,FL} = w_{2,FL} = 0.3$ ($w_{1,FR} = w_{2,FR} = 0.3$) for the left (right) steering. In Fig.6.17 the signals related to the state variables acting on the Hip 2 joints, considering the four legs, are shown when the right steering is performed. As it is possible to notice, by modulating the weight w_{ij} entries related to ipsilateral legs, the state variables signals change in amplitude producing a phase-invariant leg trajectory scaling.

6.3.4 Data-driven robot model

The development of the robot model for the MPC-based steering control was carried out using a set of transfer functions with different numbers of poles and zeros for the linear case whereas, in the nonlinear case, different I/O regressor couples were analyzed as reported in Tab.6.6. The number of hidden neurons in the neural network was fixed to 20 through a searching strategy based on a combination of expert knowledge, to fix a rough searching domain, and a grid search, to identify the best configuration in terms of prediction accuracy on the validation dataset.

The AIC analysis for all the models in Table 6.6 is shown in Fig.6.18. Therefore, the optimal model structures are 5-1 (normal friction) and 3-2 (low friction) in the linear case whereas in the neural network modelling the selected configurations are 2-4 (normal friction) and 4-5 (low friction).

In Tab.6.7 the linear and nonlinear model Fit values and the Pearson correlation coefficient R are reported using the optimal structures indicated by the AIC analysis.

Analyzing the Fit values it is evident, as expected, that the nonlinear approach outperforms the linear one. Furthermore, applying a neural network to model the robot behaviour allows for obtaining comparable performance in both scenarios with standard

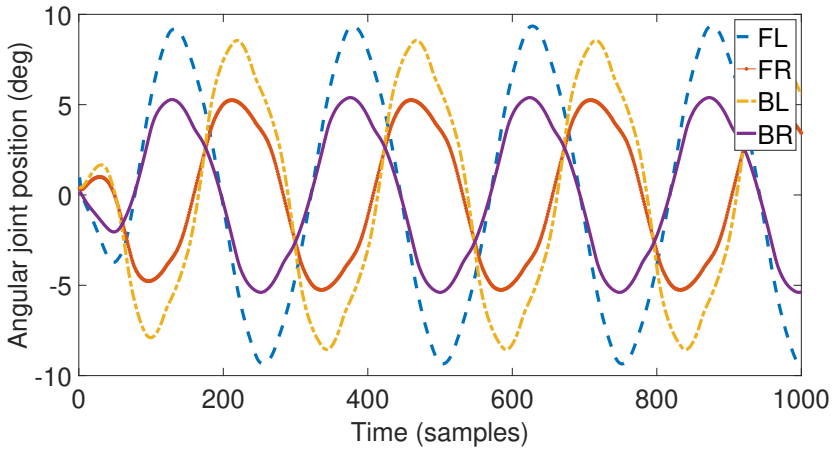


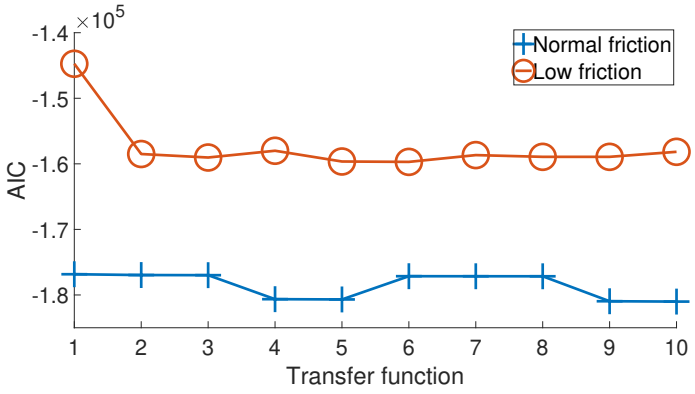
Figure 6.17: An example of the weighted phase shifted signals controlling the robot steering via the w_{ij} entries in Eq.3.2. In the legend,labels (FL, FR, BL, BR) correspond to the normalised first state variables of neuron 1,2,3 and 4,respectively, in Fig.3.1

Table 6.6: Transfer function and I/O regressors for the linear and nonlinear case, respectively. The optimal structure was selected through the AIC index.

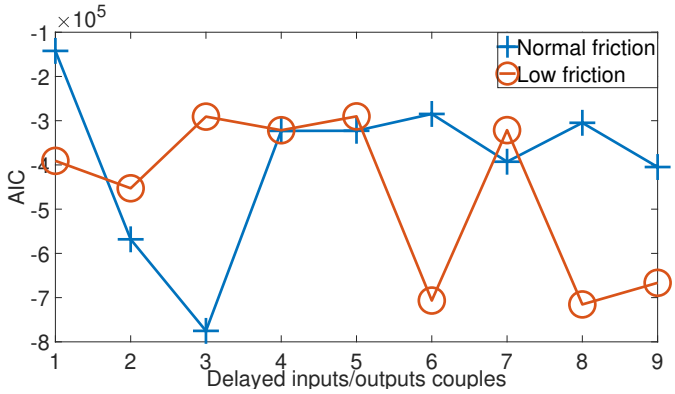
LMPC		**NNMPC**	
№	**TF poles-zeros**	**№**	**I/O regressors**
1	2-1	1	1-3
2	3-1	2	2-3
3	4-1	3	2-4
4	5-1	4	3-4
5	3-2	5	3-5
6	4-2	6	4-5
7	5-2	7	4-6
8	4-3	8	5-6
9	5-3	9	5-7
10	5-4		

and low friction, whereas the linear solution shows a significant decrease in accuracy when the low friction scenario is considered.

Fig.6.19 shows the estimation accuracy of the linear and nonlinear models for different friction conditions between the robot and the terrain. It can be noticed that the angular velocity signals obtained at low friction are less regular than in the normal friction case. However, the neural network-based model can correctly estimate the output signals in both conditions.



(a)



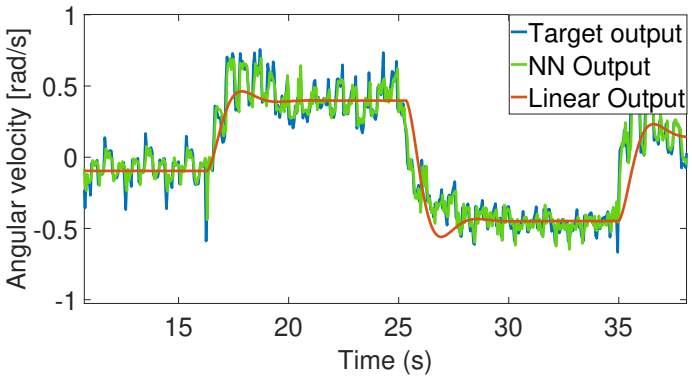
(b)

Figure 6.18: AIC index results: the linear (a) and nonlinear (b) architectures were considered in presence of normal and low friction. The numbers reported in the x-axis correspond to the configurations described in Table 6.6.

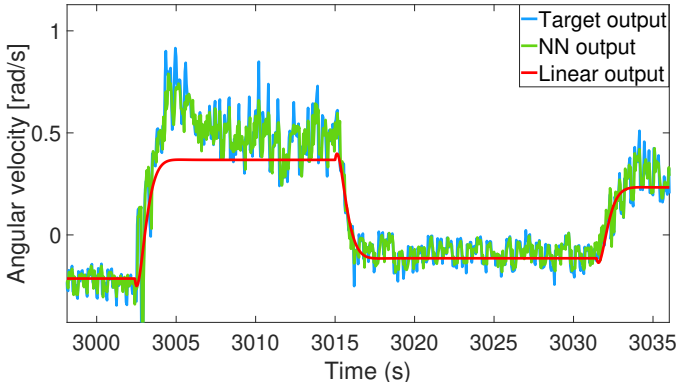
Table 6.7: Correlation coefficients and *Fit* values between the model output and the actual one for the linear and nonlinear cases in presence of normal and low friction.

Model	**Friction**	**Fit**	**R**
Linear	normal	61.67	0.91
	low	55.80	0.89
Nonlinear	normal	76.97	0.97
	low	75.10	0.97

A statistical analysis of the distributions of the error between the model output and the target signal is reported in Fig.6.20 where a typical Gaussian distribution is shown in both



(a)

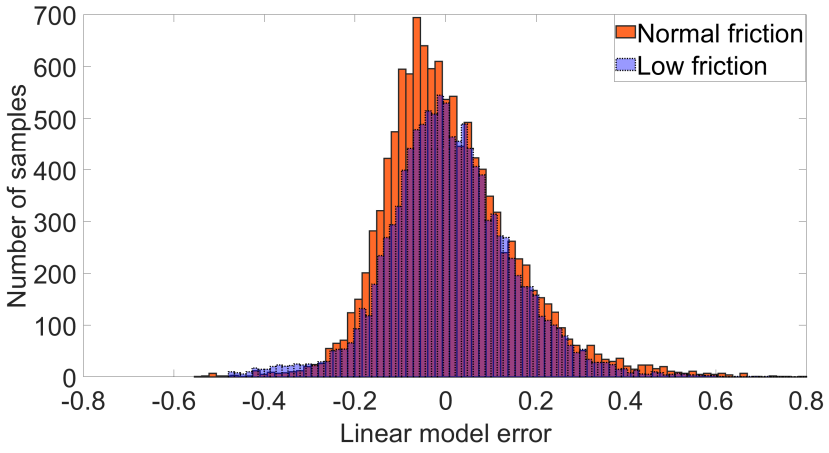


(b)

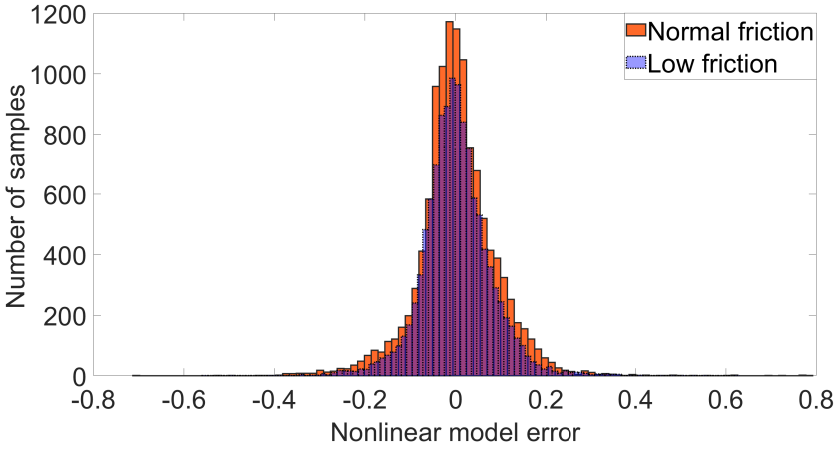
Figure 6.19: Dataset obtained with the simulation campaign in which couples gain-ω_z are collected: (a) normal and (b) low friction scenario.

cases. The improved performance of the nonlinear model is demonstrated by the reduced variance of its distribution compared to the linear approach. In the case of low friction, the larger variance seems therefore to produce larger average errors which contribute to boosting the nonlinear effects which affect the overall robot behavior. The time comparison among the models (Fig.6.19) reveals that the linear model succeeds in acquiring the average linear dynamics hidden in the nonlinear system, whereas the nonlinear model succeeds in capturing much more details of the angular velocity dynamics, typical of quadrupedal locomotion. These are instead treated as errors and filtered out by the linear model. These aspects are not detectable in the statistical plots of Fig.6.20. Moreover, the linear model seems to follow the system with a certain delay, clearly visible during the transient phases. This aspect can be negligible in the case of normal friction but can become critical in the case of low friction, where it is strictly needed to capture the high-frequency reactions of

the robot. These modelling aspects will become essential for nonlinear control success.



(a)



(b)

Figure 6.20: Statistical analysis: error distribution for the (a) linear and (b) nonlinear model in presence of normal and low friction.

6.3.5 MPC-based steering control

Three different reference signals were considered to control the robot steering: a sine wave, a sequence of steps, and a triangle reference. Fig.6.21 shows the results obtained from the dynamic simulation of the controlled quadruped robot for the different reference signals

adopting the LMPC and the NNMPC in the scenarios with normal friction. The behaviour of the robot, in presence of a more challenging scenario with low friction, is reported in Fig. 6.22.

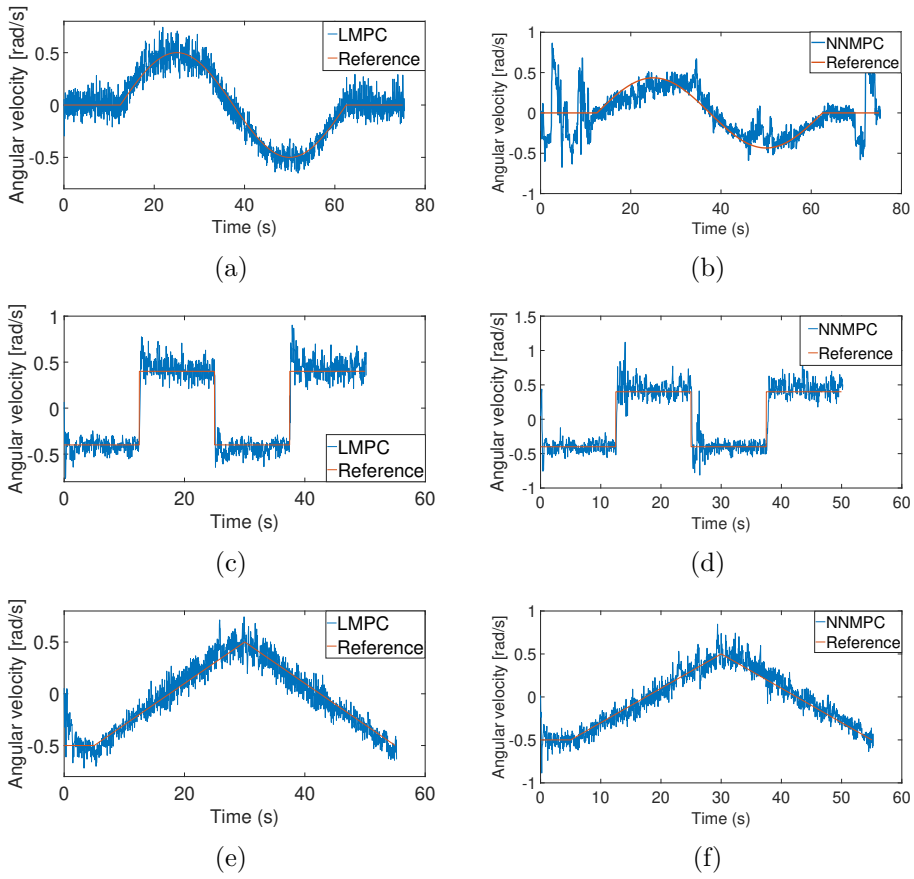


Figure 6.21: Results obtained using the LMPC and NNMPC in a normal friction condition: (a-b) sine wave reference, (c-d) step reference, (e-f) triangle reference.

As reported in Section 6.3.2, the Slipping Index was introduced as a simple method to statistically quantify the slippery effect which can cause robot failure. As already discussed, its evaluation needs the tuning of a threshold $\overline{SI}$. This can be obtained through a statistical analysis performed on the SI distribution as depicted in Fig.6.23. Here, the SI is reported for the FR leg while in the stance phase, recorded during walking. Fig.6.23(a-b) reports the values obtained using the LMPC and the NNMPC when a zero reference signal (i.e. forward path) is provided to the system. When the robot follows a simple forward trajectory, both controllers have equivalent performance. In the case of more complex reference paths, such as a sine signal the weakness of LMPC arises, as shown in Fig.6.23(d). Here, the LMPC and NNMPC controllers are compared in the low friction

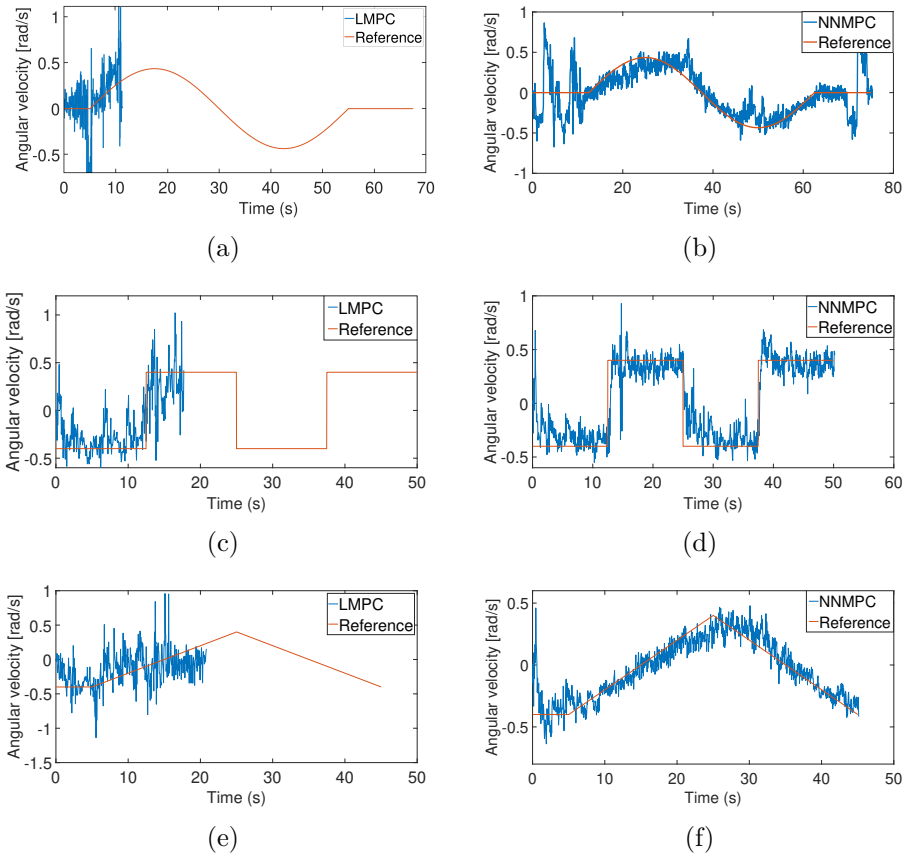


Figure 6.22: Results obtained using the LMPC and NNMPC in a low friction condition: (a-b) sine wave reference, (c-d) step reference, (e-f) triangle reference.

scenario, acquiring samples of SI soon before the falling event shown in the LMPC case (Fig.6.22(a)). From Fig.6.22(c), the SI shows a large statistical difference and is much smaller in the NNMPC application. In particular, as a result of this statistical analysis, a safe threshold to be used is $\overline{SI} = 0.02$, quantified as the boundary of the third quartile around the median SI value in the LMPC application. This can be considered as a maximum normalised speed that, if overcome, can cause the robot fall.

In presence of a typical robot ground interaction with normal friction, the performance of the NNMPC is mostly comparable with linear MPC, as reported in Tab.6.8. A normal friction parameter helps in maintaining the robot's stability while steering, reducing the need for a nonlinear approach.

The performance of the LMPC drastically degrades in presence of scenarios with low friction in the foot-ground interaction. As can be seen from Fig.6.22, in this case, when the linear model is adopted, the robot is no longer able to correctly follow the reference

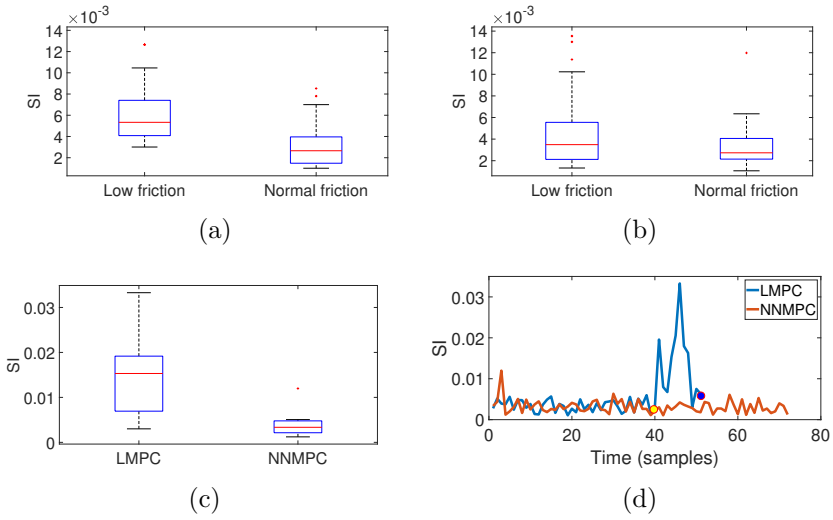


Figure 6.23: Slipping statistical analysis: (a)-(b) LMPC and NNMPC cases, respectively, when the robot follows a zero reference signal, (c) SI distribution when the robot follows a sine reference (Fig.6.22(a-b)) in the low friction scenario. (d) SI time evolution related to the previous case. The yellow circle represents the moment in which the robot starts to oscillate, the blue one when the robot is fallen. Within each box, the central mark is the median, the edges are the 25th and the 75th percentile, the whiskers extend to the most extreme data points the algorithm considers to be not outliers, and the outliers are depicted individually as '+'.

Table 6.8: *Fit* and *MSE* values obtained using the LMPC and NNMPC approaches with three different reference signals.

Ref. signal	*Fit*		*MSE*	
	LMPC	**NNMPC**	**LMPC**	**NNMPC**
Triangle	70.75	70.97	0.013	0.009
Sine wave	68.40	67.45	0.008	0.01
Steps	63.99	62.69	0.009	0.02

steering signals, falling on the ground. The NNMPC is instead able to complete the trials although the fitting between the reference and the current signal decreases if compared with the normal friction case. The performance indexes for the low friction scenario, when the NNMPC is adopted, are reported in Tab.6.9.

In Fig.6.24 the Stability and Harmony indexes are reported, showing the statistical distribution over 20 trials. Here the robot follows a straight path considering the normal and low friction environments and applying the LMPC and NNMPC control strategies.

The Stability and Harmony indexes for both the LMPC and NNMPC strategies are

Table 6.9: Fit and MSE values obtained using the NNMPC approach with three different reference signals.

Reference signal	Fit	MSE
Triangle	59.12	0.02
Sine wave	56.79	0.04
Steps	55.59	0.03

significantly better in the normal friction scenario than in the slippery terrain. The statistical relevance of the different distributions was analyzed in Tab. 6.10 using the Welch's t-test, a test decision for the null hypothesis. Considering the results, h is 1 if the test rejects the null hypothesis at the 5% significance level, and 0 otherwise. LMPC and NNMPC show the same distribution only for the stability analysis at normal friction, whereas in all the other cases the differences are statistically relevant and demonstrate the effectiveness of the NNMPC if compared with its linear counterpart.

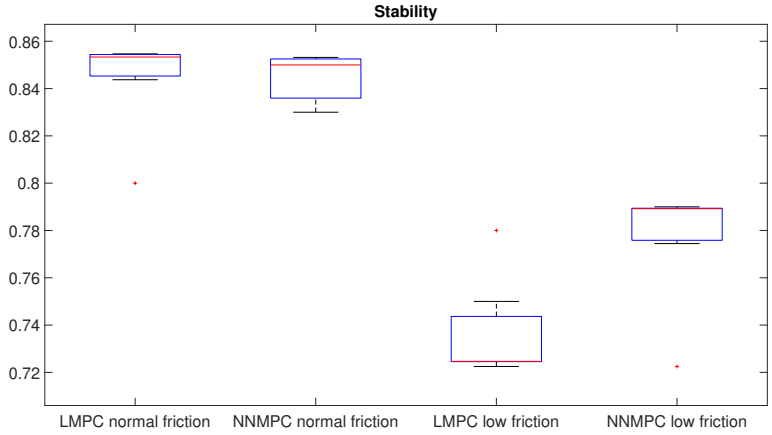
	Stability		**Harmony**	
	h	*p-value*	h	*p-value*
$\mathbf{LMPC}_{NF}$ **vs** $\mathbf{LMPC}_{LF}$	1	0.0001	1	0.000005
$\mathbf{NNMPC}_{NF}$ **vs** $\mathbf{NNMPC}_{LF}$	1	0.001	1	0.00006
$\mathbf{LMPC}_{NF}$ **vs** $\mathbf{NNMPC}_{NF}$	0	0.82	1	0.001
$\mathbf{LMPC}_{LF}$ **vs** $\mathbf{NNMPC}_{LF}$	1	0.01	1	0.04

Table 6.10: The Welch's t-test was applied to the stability and harmony indexes reported in Fig. 6.24 the subscripts indicate normal friction (NF) and low friction (LF).

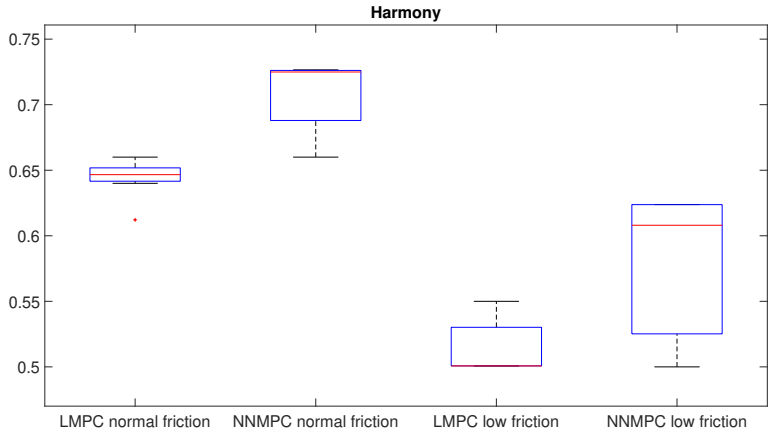
6.4 Conclusion

In this chapter, the effect of linear and nonlinear MPC-based control strategies acting on the robot steering and applied on a quadruped robot, endowed with a bio-inspired locomotion controller are analyzed. The quadruped robot behaviour has been modelled using a neural network designed and optimized following a data-driven approach. From a comparative analysis carried out considering the fit values performance and the MaxAE index, it was possible to evaluate the control results in terms of matching between the reference signals of the yaw angle of the controlled robot simulated in a dynamic framework. The comparative analysis of the results demonstrates the effectiveness of the NNMPC in particular in presence of continuously time-varying trajectories.

In the second part of the chapter, the control of the quadruped in slippery surfaces is taken into account. In particular, the low-level phase-shifted synchronization among the robot legs, for general manoeuvres including steering, was entrusted to a reaction-diffusion



(a)



(b)

Figure 6.24: Performance indexes when the robot follows a straight path considering the low friction environment applying the LMPC and NNMPC control strategies: (a) stability and (b) harmony. The statistics are performed over 20 trials, changing the robot initial leg configuration for each case.

CNN CPG. Since steering manoeuvres require a network controller using a weighted directed graph, a new theoretical result introduced in section 2 is here exploited to design an exponentially stable gait controller, imposing specific phase relations among the robot legs. Moreover, the most stable and regular gait, i.e. trot, was adopted because of the critical application of trajectory control in slippery terrains. For high-level trajectory control, linear and nonlinear (neural network-based) MPCs are compared. MPC guides the robot

steering based on a reference consisting in the angular velocity (yaw speed) and acting on specific gains modulating the neural signals applied as position control references to the robot joints. To properly compare the results, a neural network and a linear transfer function model were developed and optimized, using a data-driven approach, to model the quadruped robot behaviour. The dataset adopted was generated using a model of the Mini Cheetah robot, simulated in a dynamic environment, both in typical working conditions and in presence of slippery terrains. The results obtained with NNMPC were compared with the linear MPC-based approach. The selection of the optimal model was achieved in both cases through the AIC index, taking into account both the accuracy and the complexity of the model. A comparative analysis was carried out taking into account the Fit and the MSE indexes. The difference between the two control systems is evident in the case of relevant slippery conditions: the LMPC was no longer able to complete the requested steering trajectories causing the robot falling. The main reason is that the impulsive forces generated on the robot body during the leg touch down and stance phase can be filtered out only in the case of a normal friction condition: in this case, a linear model can approximate quite well the robot dynamic response to the steering commands. In the case of slippery surfaces, the advantage of an NNMPC is evident even if it is computationally more complex respect the LMPC. The quality of the control architecture was also demonstrated using typical indexes adopted for legged locomotion such as Stability and Harmony. These quantified the improvements obtained with an NNMPC control in the low friction case. The dynamic simulation analysis of the quadruped robot represents a needed step for future implementations on the hardware prototype. Particular attention was already devoted to reducing the computational complexity, in view of an onboard implementation, for both the developed CPG and the MPC, through the selection of the optimal linear and nonlinear robot models in terms of prediction accuracy and model complexity. The approach adopted, being essentially data-driven, can be extended to different scenarios including uneven terrains and robot architectures, as discussed.

Chapter 7

Robot-oriented neural models for path planning[1]

This chapter reports research works developed during the PON CLARA project (CLoud-plAtform and smart underground imaging for natural Risk assess-ment) whose main goal was the acquisition of a deep knowledge of the territory concerning the phenomena of hydrogeological instability and seismic risk that can affect inhabited centers. This goal has been pursued through the development of widespread smart technology that enable the acquisition, management and sharing of complex information.

In general, considering the problem related to the robotic navigation, four main aspects can be identified: *localization* (where is the robot?), *path planning* (how can the robot reach a target point from a starting point?), *motion control* (what is the better gait the robot can use considered the particular task), *perception* (how can the robot read the data coming from the sensors? and, if a sensor is damaged, how can be replaced?). In this chapter, one of the fundamental problems of mobile robot navigation is faced using neural tools, in particular, a neural-based method for the derivation of the traversability maps is proposed.

The methods for the optimal path search include rescue operations, telerobotic systems [81, 82], exploration of complex terrains, and sensor positioning landslides [5, 83]. In [83] an autonomous drilling robot for landslide monitoring is proposed, whereas in [4] the problems related to the installation and maintenance of the monitoring equipment used for landslides, owing to the complexity of the terrain, are dealt with. For instance, after a ground movement, a sensor node must either be moved or replaced to allow proper monitoring. Assessment of robot traversability is fundamental in planetary exploration

[1]The results reported in this chapter are extracted from "Learning traversability map of different robotic platforms for unstructured terrains path planning - Paolo Arena, Carmelo Fabrizio Blanco, Alessia Li Noce, Salvatore Taffara and Luca Patanè - 2020 International Joint Conference on Neural Networks (IJCNN)" and "Learning risk-mediated traversability maps in unstructured terrains navigation through robot-oriented models - Paolo Arena, Luca Patanè and Salvatore Taffara - 2021 Information Sciences."

wherein the environment and the robot have to be simulated before real tests [84]. Recently, quadruped platforms have been considered for planetary exploration [85].

The optimal paths to be followed to reach a target location based on the specific characteristic of a particular type of robot available represent the optimal solutions minimizing the path lengths while maintaining the risk associated with that path below the maximum acceptable upper bound [86, 87].

7.1 Notes on neural and shortest paths algorithms

In this section, the computing structures and algorithms adopted to obtain the results in the Chapter are briefly introduced.

7.1.1 Multilayer Perceptrons

The Multilayer Perceptron (MLP) is a Feed-forward Neural Network (FNN) consisting of elementary units called neurons connected and trained with the backpropagation algorithm associated with the Levenberg-Marquardt optimisation method. Neurons are arranged within an input layer, a hidden layer, and an output layer. The number of neurons in both the input and output layers is related to the task to be accomplished, whereas there is no fixed rule to determine the number of neurons in the hidden layer which is selected using optimisation strategies [88].

Considering a vector x including p input variables, i.e., $x = [x_1, x_2, \cdots, x_p]^T$, and a number of neurons in the hidden and output layer of q and 1, respectively, the structure $W = [w_{ij}]$ is the $p \times q$ weight matrix that connects p inputs to the q hidden layer nodes. $b = [b_h^j]$ (with $j \in [1, q]$) is a vector of biases for the hidden layer nodes. The weight $k = [k_j^O]$ ($j \in [1, q]$) is the vector that connects the q hidden layer nodes to a single output. Therefore, the input-output relation of the MLP can be expressed as follows:

$$y = g\Big(\sum_{j=1}^{q} k_j^o f\Big(\Big(\sum_{i=1}^{p} w_{ij} x_i\Big) + b_j^h\Big) + b^0\Big) \tag{7.1}$$

where $f(\cdot)$ and $g(\cdot)$ are the activation functions (e.g. sigmoidal functions) for the hidden and output layer neurons, respectively. b_j^h and b^0 are the bias parameters for the hidden and output neurons, respectively. The hyperparameters to be selected are the number of neurons in the hidden layer ($numHidden$), the neuron activation function ($actFunction$), the learning algorithm ($solver$), the maximum number of learning epochs ($MaxNumEpochs$), and the maximum number of validation failures ($MaxFail$).

7.1.2 Decision trees

A Decision Tree (DT) is a hierarchical model made up of decision rules that divide independent variables into homogeneous zones using a recursive strategy [89, 90]. DT aims to

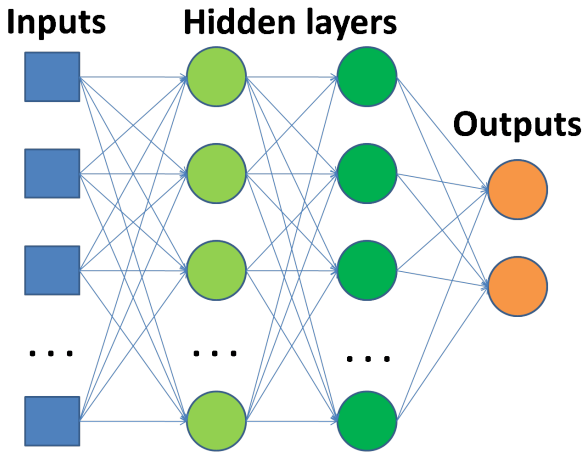


Figure 7.1: MLP neuronal structure. In the simplest structure, the MLP consists of three layers: an input, an output, and a hidden layer of nonlinearly-activating nodes. Since MLPs are fully connected, each node in one layer is connected with a certain weight to every node in the following layer.

define the set of decision rules from a set of input variables that can be used to forecast the output. If the target variables are discrete or continuous, a DT is called classification or regression tree. DT has been successfully applied for classification and estimation in many real-world contexts. The hyperparameter considered for the DTL classifier is the maximum number of splits ($MaxNumSplts$).

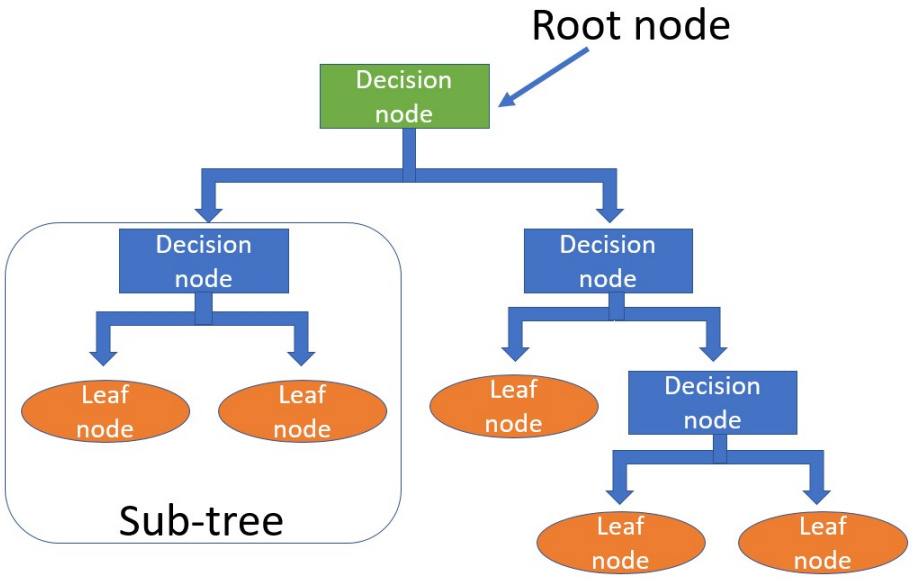
The structure of a DT is simple: the internal nodes represent the features of the dataset while the branches represent the decision rules and each leaf node describes the outcomes. In particular, there are two nodes in every decision tree: one is the decision node and the other is the leaf node.

A DT algorithm is named in this way because it starts with a root node and it expands into many branches and forms a structure like that of a tree. It simply asks a question and based on a dichotomous answer, it expands into subtrees.

To build a tree, we use the CART algorithm, which stands for Classification and Regression Tree Algorithm [91].

7.1.3 Random forest classifiers

The evolution of DTL approach to provide a more robust performance has resulted in the Random Forest (RF). This model consists of a large number of small DTs, that are trained slightly differently, each one producing its own estimation. The RF model combines these estimations to produce a more accurate prediction. The construction of an RF involves the choice of a metric for attribute collection and a pruning method [92, 93]. The training algorithm for RF applies the general technique of bagging [94] to tree learners. One decision tree is trained alone on the whole training set. In a RF, N DTs are trained each



(a)

Figure 7.2: A decision tree learning algorithm structure. It starts with a root node and it expands into many branches and forms a structure like that of a tree. It simply asks a question and based on a dichotomous answer, it expands into subtrees.

one on a subset of the original training set obtained via bootstrapping [95] of the original dataset, i.e., via random sampling with replacement. The hyperparameter considered for the RF classifier, in addition to that used for the DT, is the number of DTs in the ensemble ($numTrees$).

7.1.4 Dijkstra's algorithm

Given a positive, weighted, and directed graph $G = (T, E)$, containing T nodes and E edges, for any node s, the Dijkstra's algorithm finds the path with the lowest cost (i.e. shortest path) between s and the other nodes in G. It can also be used to find the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined [96]. In particular, the Dijkstra's algorithm is applied to directed graphs evaluated with non-negative evaluation, i.e., characterized by non-negative arcs.

It is structured as follows:

D1. A coefficient λ_i is associated to each node of the graph so that $\lambda_1 = 0$ and $\lambda_i = \beta_{1i}$ $(i = 2, ..., n)$ with β_{1i} the valuation of the arc (x_1, x_i) (if this arc does not exist,

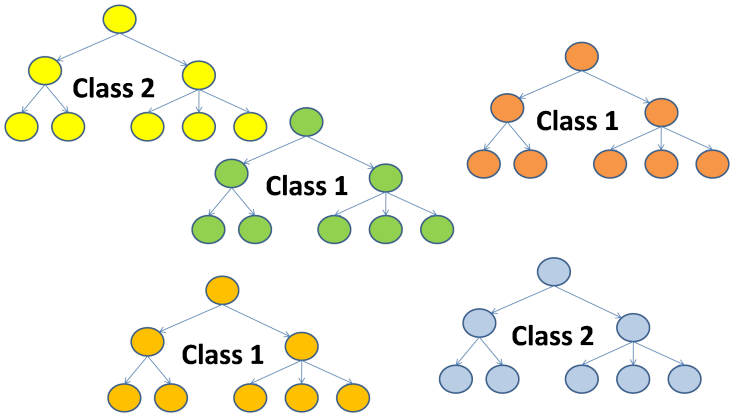


Figure 7.3: RF structure: the algorithm relies on multiple DTs that are all trained slightly differently, all of them are taken into consideration for the final classification.

then $\beta_{1i} = +\infty$). The λ_1 coefficient is declared *definitive* and it is indicated with λ_1^*. The algorithm goes to step D2.

D2. If all λ_i $(i = 1, ..., n)$ coefficients were declared *definitive*, the algorithm stops, otherwise it goes to D3.

D3. Let λ_n^* the last coefficient declared *definitive*. Each temporary coefficient λ_i is replaced with $\lambda_i = min\{\lambda_i, \lambda_n^* + \beta_{ni}\}$. Among the new temporary coefficients λ_i, the minimum value one is chosen and it is declared as *definitive*. The algorithm goes to D2.

An example of how the Dijkstra's algorithm works is shown below, the related graph is shown in Fig.7.4.

Start

First step:

$\lambda_1^* = 0$, $\lambda_2 = 13$, $\lambda_3 = \lambda_4 = \lambda_5 = +\infty$, $\lambda_6 = 8$, $\lambda_7 = 12$.

Second step:

$\lambda_2 = min\{\lambda_2, \lambda_1^* + \beta_{12}\} = min\{13, 0 + 13\} = 13$

$\lambda_3 = min\{+\infty, 0 + \beta_{13}\} = min\{+\infty, +\infty\} = +\infty$

$\lambda_4 = min\{+\infty, 0 + \beta_{14}\} = min\{+\infty, +\infty\} = +\infty$

$\lambda_5 = min\{+\infty, +\infty\} = +\infty$

$\lambda_6^* = min\{8, 0 + 8\} = 8$. This is declared *definitive*.

$\lambda_7 = min\{12, 0 + 12\} = 12$

Third step:

$\lambda_2 = min\{13, \lambda_6^* + \beta_{62}\} = min\{13, 8 + \infty\} = 13$

$\lambda_3 = min\{+\infty, 8 + \beta_{63}\} = min\{+\infty, +\infty\} = +\infty$

$\lambda_4 = min\{+\infty, 8 + \beta_{64}\} = min\{+\infty, +\infty\} = +\infty$

$\lambda_5 = min\{+\infty, 8 + \beta_{65}\} = min\{+\infty, 8 + 11\} = 19$

$\lambda_7^* = min\{12, 8 + \beta_{67}\} = min\{12, 8 + \infty\} = 12$. This is declared *definitive*.

Fourth step:

$\lambda_2^* = min\{13, 12 + \beta_{72}\} = min\{13, +\infty\} = 13$. This is declared *definitive*.

$\lambda_3 = min\{+\infty, 12 + \beta_{73}\} = +\infty$

$\lambda_4 = min\{+\infty, 12 + 15\} = 27$
$\lambda_5 = min\{19, 12 + 15\} = 19$
End

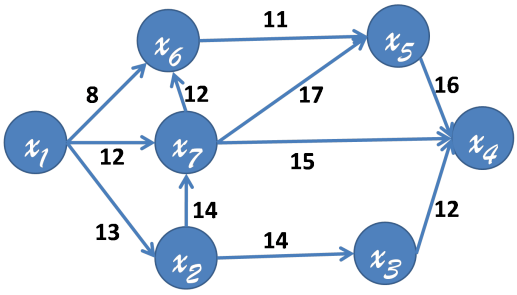


Figure 7.4: Example of graph in which Dijkstra's algorithm is applied.

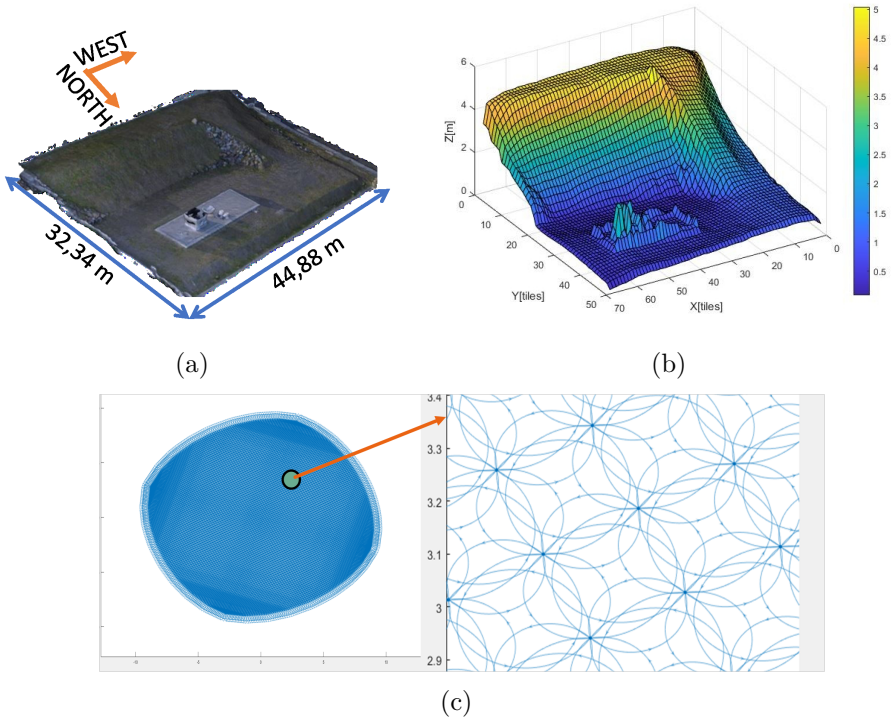
7.2 Terrain mathematical formalization

To provide an exhaustive methodology for the generation of traversability maps, it is useful to provide a mathematical formalism related to the basic elements constituting the traversability problem. Considering a real terrain map shown in Fig.7.5(a), the heightmaps is shown in Fig.7.5(b) and the related locally connected (each node is connected to its neighboring nodes), fully oriented (each arc has a direction) graph in Fig.7.5(c).

- $\mathcal{C}$ is the point cloud generating the terrain map and it is a matrix of uniformly spaced height points after a downsampling that generated a set $\mathcal{P} \subset \mathcal{C}$.
- $G = (T, E)$ is the graph connecting the points in $\mathcal{C}$ and is characterised by regular tessellation (Fig. 7.5(b)), where:
  - $T = \{T_{i,j}\}$ is the set of tiles (i.e. nodes of the graph), here represented through quadrangles of fixed dimension T_d and corners identified through their 2D coordinates $T_{i,j} = \{p_{i,j}^{++}, p_{i,j}^{+-}, p_{i,j}^{-+}, p_{i,j}^{--}\}$, where each element contains the elevation of the corresponding point in the map.
  - $E = \{E_{i,j}\}$ is the set of edges representing the paths between the centre of the considered tile and its neighbours in the four cardinal and diagonal directions: $E_{i,j} = (e_{i,j,i-1,j-1}, e_{i,j,i-1,j}, \cdots e_{i,j,i+1,j}, e_{i,j,i+1,j+1})$. In Fig. 7.6, a description of a generic tile $T_{i,j}$ and its constituent elements is reported.
- $V_{k,l \to m,n}$ is the path connecting two generic tiles $T_{k,l}$ and $T_{m,n}$, and it is described by an ordered sequence of adjacent tiles along the corresponding edges, characterised by the starting and ending edges $V_{k,l \to m,n} = \{e_{k,l,*,*}, \cdots e_{*,*,m,n}\}$.

- $\mathcal{L}(V_{k,l\to m,n})$ is the length of a path and it is defined as the number of edges contained in that path. The shortest path $\overline{V}_{k,l\to m,n}$, whose length is $\mathcal{L}(\overline{V}_{k,l\to m,n})$ is defined as that characterised by the lowest number of edges among all the other paths $V_{k,l\to m,n}$.

The previous indexes are reported in Fig.7.6 describing the constituting elements of a generic tile $T_{i,j}$.



(a) (b)

(c)

Figure 7.5: Terrain used as testbed: (a) 3D reconstruction; (b) 3D height map; (c) locally connected, fully oriented graph. The zoom of a detail is shown.

7.3 Robot-oriented neural model: the algorithm

The proposed algorithm for terrestrial navigation is summarised in Fig.7.7. It is divided into two parts: the learning process and the model exploitation. In the first one, the robot-oriented neural models are generated starting from a training dataset. In the second one, the model obtained in the previous part is used to obtain the traversability maps and, consequently, the optimal paths, based on different approaches.

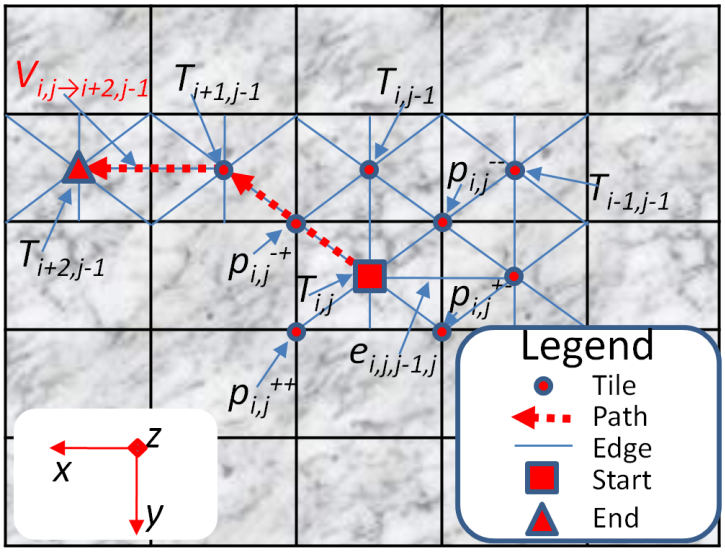


Figure 7.6: Description of a generic tile $T_{i,j}$ and its constituting elements. A path $V_{i,j\to i+2,j-1}$ is reported.

Learning process

1. **Training data.** The first step involves the selection of a training map from among a set of target maps.

2. **Morphological features extraction and dynamic simulations.** The morphological characteristics are extracted after partitioning the terrain into tiles of fixed dimensions. In parallel, a set of selected robots was simulated in a dynamic environment containing the training map. The robots taken into account, as it is possible to see in the flow chart, are four, i.e.:

   (a) Robotnik Summit-XL (upper right) that is a four-wheeled platform with skid-steering kinematics in which each wheel integrates a brushless motor with a gearbox and encoder mounted on an independent suspension system [97].

   (b) 5BSPL (upper left), a quadruped-like robot adopting a kinematic structure composed of a set of five links connected in a closed chain by five revolute joints, two of which are actuated by brushless DC electric motors (BLDC) and two are passive, made up of ball bearings. This structure is called "5-bar symmetric planar linkage" [98].

   (c) Asguard robot (bottom right) is a hybrid prototype with an interesting structure composed of four rotating wheels modelled as a five-pointed star [99]. In this work, the robot is controlled using a bio-inspired Central Pattern Generator (CPG) controller. In [100], an overview of the relationship between locomotion and neural control mechanisms is given. This robot is highly agile

and fast on flat ground and, at the same time, it is capable of dealing with very rough terrains such as rubble, gravel, and even stairs.

(d) Cooperative Asguard (bottom left), it is a structure created to improve the locomotion capabilities of the Asguard robot, following a bio-inspired strategy developing a coupling mechanism between two robots. Inspiration comes from snails and octopuses that, to attract females, perform a courtship ritual aimed at creating a mechanical connection with the partners. Nature can provide simple solutions that can be technically reproduced and adapted for other purposes. The male robot has a harpoon with two actuated compliant joints, one oriented along the horizontal axis and the other along the vertical axis. On the other hand, the female robot has a funnel in the front part, in which the harpoon can be inserted and locked mechanically. The male-actuated appendage facilitates assembly owing to the difficulties that occur when the alignment procedure is performed on uneven terrains. Through a searching algorithm, the male robot can approach and mechanically couple with a female. The harpoon joints are actuated only during the search and coupling phases. After that, they are no longer actuated, and the two-robot link behaves as a two-DoF spring damped passive coupling system. In this study, already assembled robots will be considered. The expected advantages of a coupled structure are improved stability and the capability to overcome higher obstacles than a single system.

Each robot could move from one tile to a neighbour, randomly selected to evaluate its capabilities. For the selected tiles, a series of morphological features were collected and used as input for the neural network, whereas the network targets encoded the success or failure of the robot in traversing the tiles in a given direction.

3. **Robot-oriented neural models.** The developed dataset is then provided to a series of neural networks to learn the specific roving robot capabilities in relation to the terrain configurations. The robot-oriented neural model, one for each robot, is obtained.

Model exploitation

1. **Target data selection.** The first step of the model exploitation part involves the selection of the target map, i.e., the map for which the optimal paths have to be obtained.

2. **Morphological features extraction.** From the selected target map, the dataset related to the morphological features is obtained.

3. **Robot-oriented neural models.** The neural network trained in the previous learning process is used taking in input the dataset generated in the previous step to obtain the robot-oriented neural models.

4. **Direction-based traversability maps.** The learned models are subsequently tested on different terrains (i.e., target maps), providing traversability maps in real-time without the need to run the simulator. This strategy was applied to

generate eight traversability maps (four cardinal and four diagonal directions) for each robot.

5. **Optimal path algorithm.** The traversability maps are used as inputs for a path-planning algorithm to provide the best robotic structure to be employed and the optimal route between specific locations identified on the map (i.e., assigned task) depending on the level of the admissible risk selected, as clarified in the flowchart and in the following sections.
6. **Optimal path validation on dynamic simulator.** The results obtained in the previous steps are verified in the dynamic simulator.

7.3.1 Morphological features extraction

The target terrain used to test the methodology was acquired by a drone with an onboard camera. The reconstructed height map has a spatial resolution of approximately 13 cm in the x and y directions and $1cm$ for the altitude (z-direction)[101]. To allow a compromise between fast data processing and the average size of the adopted robots, the map was downsampled and the final spatial resolution was fixed to 65 $cm/tile$ for the evaluation of the optimal paths. This allows the maintenance of some height points per tile equal to $5*5$, which is useful for a suitable characterisation of the tile roughness and the maximum heights that our robots can encounter, as discussed below.

The method adopted to obtain the morphological characteristics of the terrain belongs to the class of geometric approaches [102]. The height map of the terrain, as shown in Fig. 7.5(b), is used to extract the most salient morphological characteristics embedded in each tile: average height difference, maximum height, average slope, and average roughness.

1. **Average height difference.** If $T_{i,j}$ and $T_{m,n}$ are two consecutive tiles, the heights are $H_{T_{i,j}} = mean(p_{i,j}^{++}, p_{i,j}^{+-}, p_{i,j}^{-+}, p_{i,j}^{--})$ and $H_{T_{m,n}} = mean(p_{m,n}^{++}, p_{m,n}^{+-}, p_{m,n}^{-+}, p_{m,n}^{--})$, respectively. The height difference between the two tiles is $\Delta H_{T_{i,j},T_{m,n}} = H_{T_{i,j}} - H_{T_{m,n}}$.
2. **Maximum height.** The maximum height that the robot has to face while moving through the considered tile is $H_{T_{i,j}}^{max} = max(\mathcal{C}|_{T_{i,j}})$.
3. **Average slope.** It is related to the motion direction. Considered $T_{i,j}$ and a robot moving from the south to the north direction (i.e. y-axis direction) on it, the slope is $S_{T_{i,j}} = \arcsin(\frac{1}{T_d}(\frac{p_{i,j}^{++}+p_{i,j}^{+-}}{2} - \frac{p_{i,j}^{+-}+p_{i,j}^{--}}{2}))$.
4. **Average roughness.** For each tile $T_{i,j}$, we have a cloud of point heights $\mathcal{C}|_{i,j}$ extracted from the original heightmap before downsampling. The least-squares plane fitting method was used. An interpolation plane limited to the considered tile points $\Pi|_{T_{i,j}}$, as shown in Fig.7.8(b), was created. The average distance of the points ($d_{k,l}$ for the generic $c_{k,l}$ point) in $\mathcal{C}|_{T_{i,j}}$ from the plane $\Pi|_{T_{i,j}}$ represents the tile roughness $D_{T_{i,j}} = mean(dist(\mathcal{C}|_{T_{i,j}}, \Pi|_{T_{i,j}}))$.

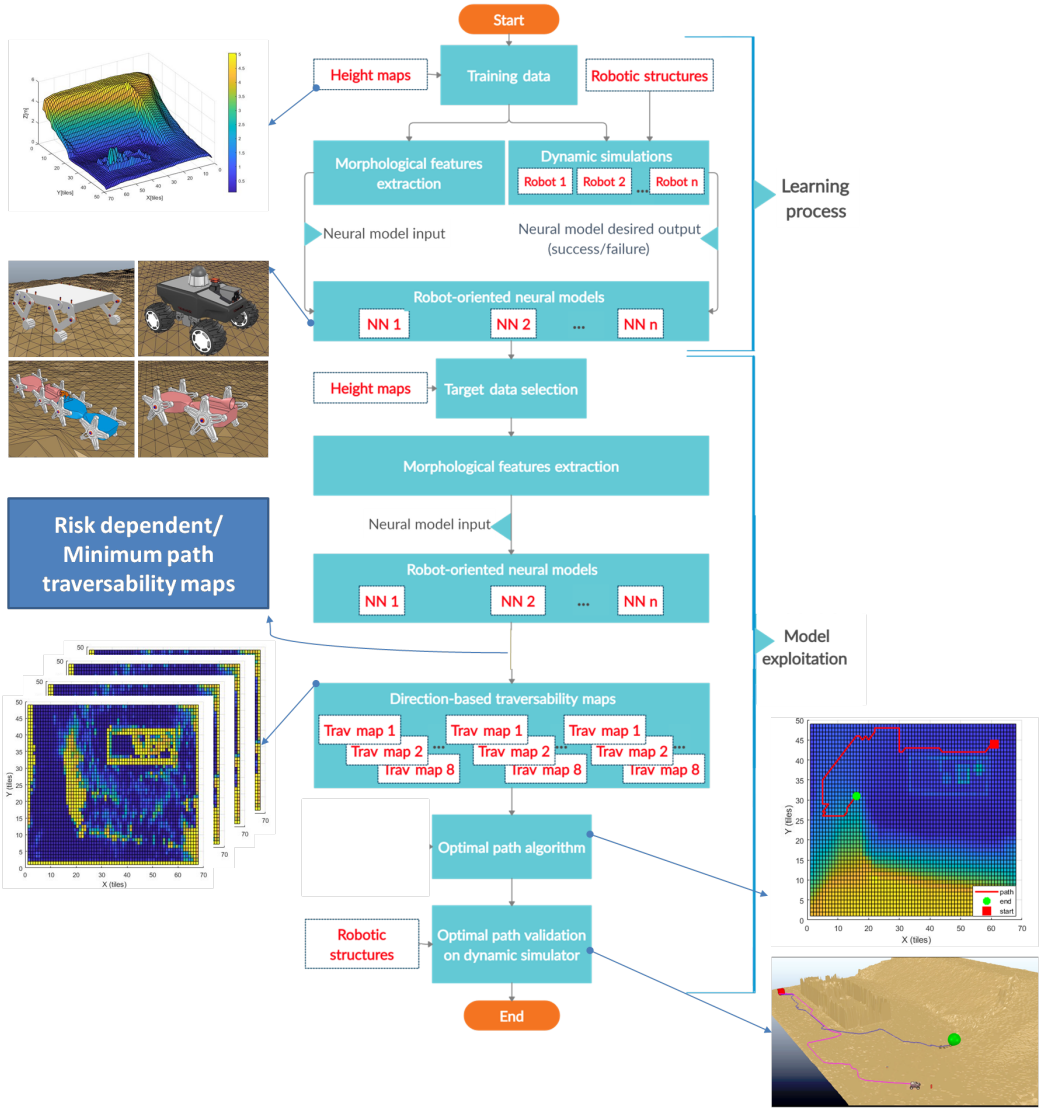
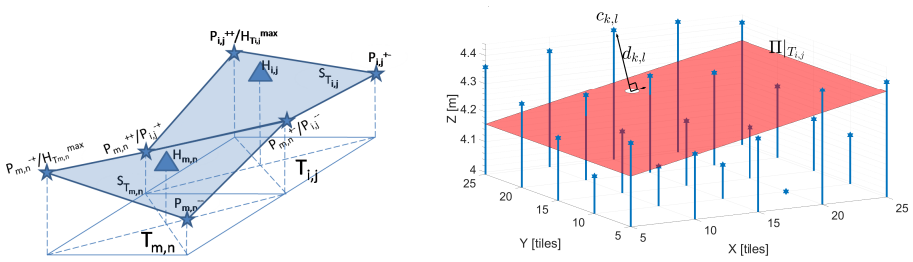


Figure 7.7: Flow chart describing the entire algorithm that leads to the neural models generation.

The previous indexes can be visualized in Fig.7.8. In particular, in Fig.7.8(a) the average heights, maximum heights and average slopes of the tiles $T_{i,j}$ and $T_{m,n}$ are reported while in Fig.7.8(b) the roughness of a set of $5 * 5$ tiles is represented.



(a) (b)

Figure 7.8: Morphological parameters representation: (a) two neighbor tiles $T_{i,j}$ and $T_{m,n}$ and the relative parameters: $H_{T_{i,j}}$ (height), $S_{T_{i,j}}$ (slope), $D_{T_{i,j}}$ (roughness), $H^{max}_{T_{i,j}}$ (maximum height); (b) Distribution of a matrix of 5x5 height points constituting a generic tile and the corresponding interpolation plane. $R_{T_{i,j}}$ is calculated by mediating the distances of each point in the tile and interpolation plane. The triangles (in (a)) represent the tile heights while the blue stars (in (a) and (b)) represent the height points.

7.4 Dataset generation

To obtain the dataset used to train the neural networks and to obtain the traversability maps, the selected robot is left to freely explore the selected environments and the success or failure in traversing a given area is detected. A neural model, for each vehicle, is learned to acquire traversing path-specific traversability skills. The traversability information is binary coded:

1. 0 if the robot can pass the tile,
2. 1 otherwise.

Firstly, the traversability information is collected only considering the cardinal directions, then, exploiting the generalisation capabilities of neural structures, the diagonal directions can be obtained and the models can be adapted for other terrains (Fig.7.9).

The direction control of the robot is implemented using a PI controller acting on the position and orientation errors between the robot's centre of mass and the next target point to be reached along the path which is divided into a certain number of waypoints. The control acts on the wheel rotation direction or on the leg trajectories, allowing the robot to steer when necessary, based on a proportional-integral (PI) controller.

The training dataset is composed of more than 2000 patterns for each robot, divided into learning (65%), validation (15%), and test data (20%), randomly selected from the simulation environment. The hyperparameters used to set the different structures are here reported:

MLP

1. $numHidden$= [1 100]
2. $actFunction$= tanh (hyperbolic tangent)

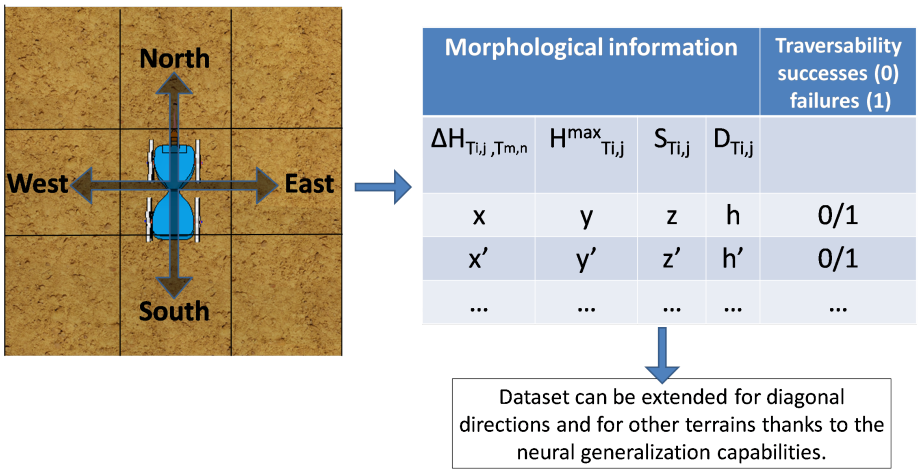


Figure 7.9: Schematization of the process through which the dataset is obtained. Starting from the terrain morphological features and the successes/failures in traversing the tiles forming the terrain considering the cardinal directions, the dataset is built. The diagonal directions can be obtained using the neural generalization capabilities.

3. *solver*= sgd (stochastic gradient descent)
4. $MaxNumEpochs$= 1000
5. $MaxFail$= 6 (training stops if validation performance has increased more than $MaxFail$ times since the last time it decreased)

DTL

1. $MaxNumSplits$= 100, and for RF $numTrees$=1000.

RF

1. $numTrees$=1000.

7.4.1 Performance Predictive evaluation

A simulation campaign was conducted to evaluate the accuracy, sensitivity, and specificity of the models.

In the MLP case, as shown in Fig.7.10 the accuracy values for the test phase, when the number of neurons in the hidden layer changes from 1 to 100 for each robot are reported. To obtain a reliable statistical value, for each network configuration, 100 learned networks were trained, each time with different randomisations between learning and test patterns. For the DTL method, the accuracy is related to the maximum number of splits of the tree. Fig.7.11(a) shows the accuracy distribution during the test phase for the Cooperative Asguard. Fig.7.11(b) shows the trends of the curves representing the accuracy of RF

changing the number of trees selected, applying the method for the same robot. From Figs. 7.10 and 7.11, the optimal parameter selection obtained using a grid searching strategy, is reported for each method.

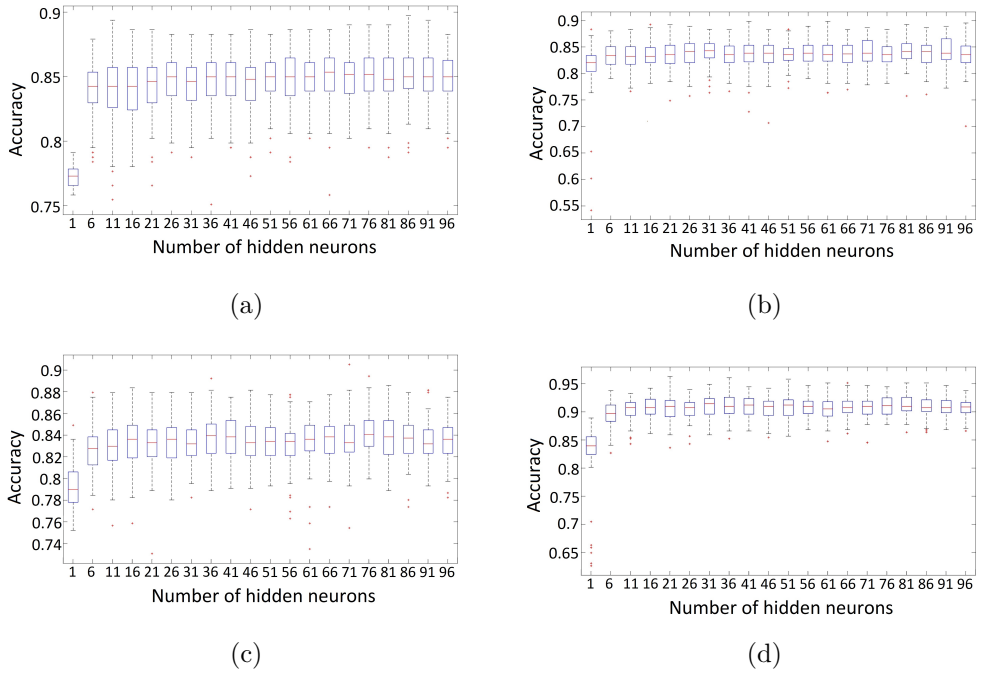


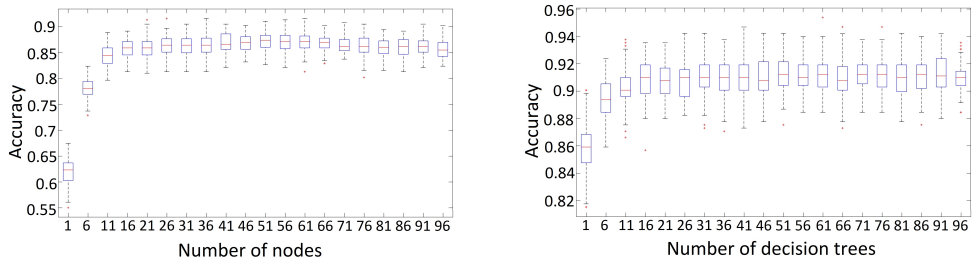
Figure 7.10: Accuracy results for the MLP in testing for each robot: (a) Robotnik; (b) 5BSPL; (c) Asguard; (d) Cooperative Asguard. The statistics were performed over 100 trials, changing the network initialisation as well as learning and test data splitting for each case. Within each box, the central mark is the median, the edges are the 25th and the 75th percentile, the whiskers extend to the most extreme data, the algorithm is considered to be not an outlier, and the outliers are depicted individually as +.

7.5 Traversability maps estimation methods

Using the learned models, it is possible to generate the traversability maps for the eight considered directions by selecting any real terrain map used as a testbed for which a heightmap is available.

In Fig.7.12, the three different strategies used for the traversability map generation are schematized considering also the relations between them. The three methods are:

- Minimum path traversability maps
- Risk-dependent traversability maps



(a) (b)

Figure 7.11: Test accuracy distribution for the Cooperative Asguard robot: (a) DTL; (b) RF.

Table 7.1: Performances indexes obtained on the test phase for the MLP, DT, and RF methods applied to the different robots. The indexes are accuracy, sensitivity, specificity, and F1-score. The optimal path parameters for each configuration are also reported: the number of hidden layers for the MLP, the number of nodes for the DTL, and the maximum number of tree splits for the RF.

Robots	Arch. type	Accuracy	Sensitivity	Specificity	F1-score	Opt. param.
Robotnik	MLP	$87.93 \pm 2.11\%$	$86.34 \pm 2.84\%$	$83.09 \pm 3.03\%$	84.68	10
	DT	$85.39 \pm 2.07\%$	$89.52 \pm 3.11\%$	$82.18 \pm 3.55\%$	85.69	17
	RF	$89.18 \pm 1.70\%$	$89.78 \pm 2.36\%$	$88.65 \pm 2.52\%$	89.21	89
5BSPL	MLP	$84.97 \pm 2.51\%$	$91.45 \pm 4.10\%$	$78 \pm 3.57\%$	84.19	69
	DT	$80.30 \pm 2.19\%$	$84.14 \pm 3.21\%$	$76.30 \pm 3.55\%$	80.02	72
	RF	$83.20 \pm 1.81\%$	$85.90 \pm 2.72\%$	$80.88 \pm 2.65\%$	83.31	83
Asguard	MLP	$84.90 \pm 1.99\%$	$94.35 \pm 2.19\%$	$78.92 \pm 2.79\%$	85.94	43
	DT	$86.77 \pm 1.73\%$	$91.44 \pm 1.97\%$	$79.73 \pm 3.34\%$	85.18	36
	RF	$87.14 \pm 1.44\%$	$89.35 \pm 1.82\%$	$85.15 \pm 2.18\%$	87.19	100
Coop. Asg.	MLP	$90.89 \pm 2.01\%$	$95.11 \pm 1.77\%$	$84.55 \pm 2.96\%$	89.51	75
	DT	$86.70 \pm 1.79\%$	$90.81 \pm 2.03\%$	$80.32 \pm 3.55\%$	85.24	58
	RF	$91.32 \pm 1.17\%$	$92.38 \pm 2.01\%$	$90.33 \pm 2.01\%$	91.34	89

- Minimum path length and larger risk traversability maps.
- Minimum risk and longer path length traversability maps.

In particular, the minimum path traversability maps take into account the robot's safety showing all the paths that the robot can traverse without harm, the risk-dependent traversability maps [86] provide traversability maps subject to an acceptable risk.

The risk-dependent traversability maps and the minimum path traversability maps methods are obtained from the traversability maps built considering the average of the

thresholded outputs from the 100 MLP structures previously learned. The obtained result provides a continuous value between 0 and 1 where the extreme conditions indicate that the considered tile is either easily traversable or impossible to pass through, whereas the values in between are associated with the riskiness of carrying out the action.

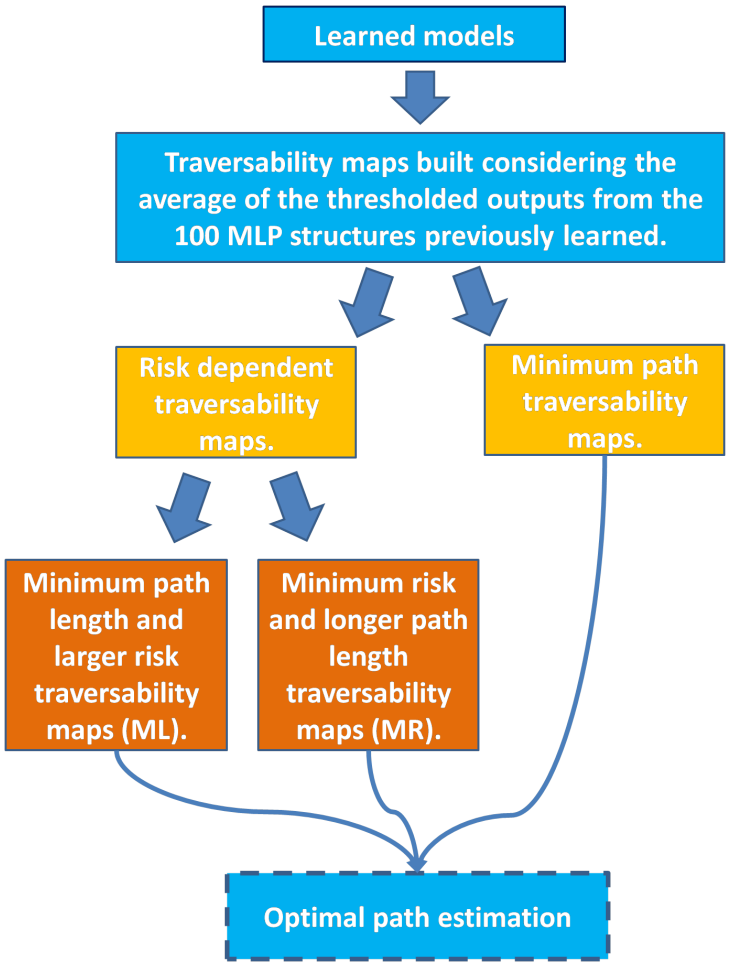


Figure 7.12: Traversability maps estimation methods: minimum path and risk dependent traversability maps.

7.5.1 Minimum path traversability maps

Directional traversability maps are built such that each element represents a physical point in the map, and its value is related to the possibility for the robot to reach that point coming from the opposite direction. This strategy consists of comparing each MLP output

that belongs to the range $[0, 1]$ to a given threshold $Th = 0.5$. This binary discriminates whether the neural model estimates the robot as successful in traversing that specific tile. We assign a value $M_{max}(M_{min})$ if the output is above (below) the threshold and compute the average value for the 100 MLP previously learned.

From the averaged outputs of the neural networks, the eight directional traversability matrices M^* (where * spans all four cardinal and four diagonal directions) are created through the following procedure: each element in the matrix represents a tile and its value is provided by the corresponding neural-network-averaged output. It quantifies the possibility for the robot to reach, from the chosen tile to the next tile, moving in the considered direction. This defines the value of the directed edge $e_{i,j;*,*}$ for tile $T_{i,j}$. For example, in the case of the north traversability matrix M^N, $M^N(i,j) = e_{i,j;i,j+1}$ (as depicted in Fig.7.6). $M^N(i,j) = M_{min}$ ($M^N(i,j) = M_{max}$), indicates that, from $T_{i,j}$, the robot is completely able (unable) to reach the neighbouring north tile $T_{i,j+1}$. This strategy was repeated for each of the eight maps.

In Fig. 7.13, the north traversability map related to the Asguard robot is reported as an example. They refer to the terrain used to learn the neural networks even if only a small amount of data was used during the training phase (approximately $2 \cdot 10^3$ patterns over the $26 \cdot 10^3$ reported in the entire traversability maps). The Asguard robot cannot pass through high-value areas, whereas others can be travelled more safely. A comparison of the obtained results with the terrain reconstruction is shown in Fig.7.5; it is clear that traversability maps are a visual representation of the robot's motion capabilities within the environment. The results show that the learned models can be usefully applied to general terrain configurations, thus representing an internal model that implicitly embeds the robot features (including kinematics and dynamic constraints) useful for solving the traversability problem.

7.5.2 Risk dependent traversability maps

This possibility of modifying the risk level when travelling on the selected path information can be considered by considering the average outcome of the 100 MLPs. This value was allowed to vary from approximately 0 (very low-risk/easy path) to 1 (very high-risk/non-traversable path). To assign this "level of risk" to each directed edge, a maximum tolerable risk (R_{max}) is defined; all the averaged MLP outputs larger (smaller) than R_{max} will be saturated to the maximum value M_{max} (minimum value M_{min}) to make the corresponding links non-traversable (traversable). Therefore, we recreate the directed edge values as described before (i.e., the matrix M^* for each direction) to obtain the directed traversability maps. The following optimal path planner extracts the path (if it exists) from two selected points with minimal length and is subject to a risk below R_{max}. As introduced before, this strategy will be therefore called "risk-dependent traversability map with minimum path length and larger risk" (ML). Another possible strategy consists of selecting the R_{max} in searching not for the shortest path, but for the least risky one; this is implemented by fixing the edge values. This strategy is called "risk-dependent traversability map with minimum risk and longer length" (MR).

To analyse the outcome of these possible strategies, a series of different scenarios are analysed in the following, referring to the Pareto optimality [103, 104, 105].

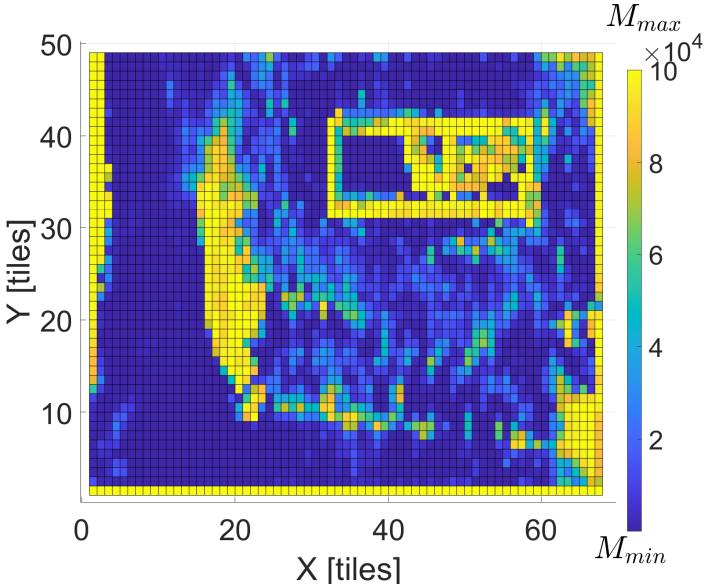
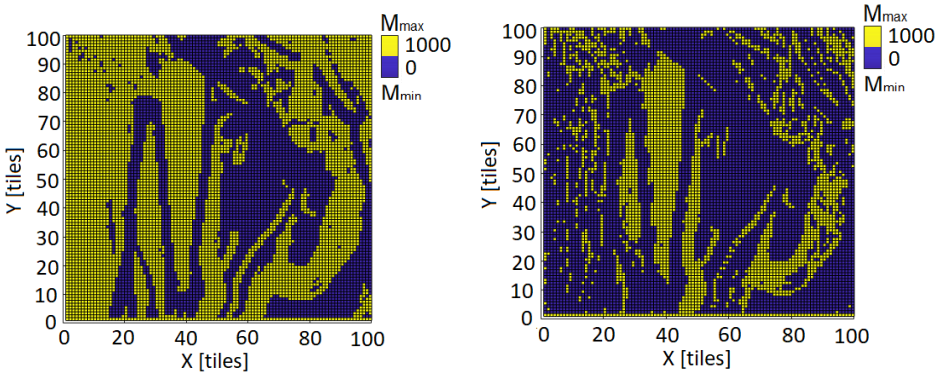


Figure 7.13: Traversability maps obtained for the Asguard robot in the north direction using the minimum path traversability map method. The extreme values $M_{min}(M_{max})$ indicate that the robot is always able (unable) to traverse the considered area in a given direction. Values between these extreme cases indicate different levels of success.

The difference between these methods is related to the fact that in ML the traversability matrix ($M^*_{i,j}$) contains continuous values depending on the averaged MLP models, and the latter (MR), in which $M^*_{i,j}$ contains binary values that account for the maximum risk chosen to minimise the path length and then further extended to identify the minimum risky path.

To obtain the thresholded maps, the following steps are performed as described above; once the non-thresholded maps are obtained, a risk threshold R_{max} is applied. The map values lower than R_{max} are set at M_{min}; those larger or equal to this are set to M_{max}. For each robot, different traversability maps can be obtained; for instance, setting a threshold $R_{max} = 0.1$ generates an optimal path that is safer than that generated with $R_{max} = 0.9$. Fig.7.14 shows the northward traversability maps for the Asguard robot taken as an example, considering these two extreme thresholds: 0.1 and 0.9. This strategy allows obtaining binary images which show the traversability maps according to the level of risk imposed. A higher risk level leads to an increase in the number of possible paths and, consequently, the possibility of finding shorter lengths for the final path.



(a) (b)

Figure 7.14: North traversability maps with $R_{max} = 0.1$ (left column) and $R_{max} = 0.9$ (right column) for the Asguard robot.

7.6 Optimal path estimation

Considering the three previously described methods, it is possible to derive the optimal paths by selecting a given robot and a different terrain with respect to the one used to obtain the neural model using the generalization capability of neural networks. Fig.7.15 shows the optimal path obtained for the Asguard robot using the minimum path traversability maps method, while Fig. 7.16 (a-b) shows the paths obtained using the ML and MR approaches. Figure 7.16 (c) shows the shortest path taken by the robot using the MR method. The robot used in the simulations in Fig. 7.16 is the 5BSPL.

A motivation that could lead to choosing either the shortest, but the risky path (ML), or a less dangerous but longer path (MR), could be driven by a time-saving priority (first case) or by a safety-preserving priority (second case). A possible side effect of choosing a risky path is related to the stress of the actuation module. In fact, the overall torque requested from the robot actuators is, on average, larger when the robot travels steeper and more complex roads. Moreover, choosing a risky path implies that it is sometimes unable to complete the assigned path. This concept is supported by the results reported in Table 7.2, which compares the average torques of the four robots considering the short and long paths.

A strategy to select minimum risk paths consists of modifying the structure that creates the previously introduced thresholded traversability maps. Once a certain threshold value is set, the values of the non-thresholded map that are larger than the latter are saturated to the value of M_{max}, whereas the lower ones maintain their own values. Thus, the average weight from the MLP averaged output below M_{max} represents the average of a failing traversability. This leads the algorithm to find the optimal path in terms of minimum cost tiles, but not necessarily the shortest in terms of the number of tiles travelled compared with the previous strategy; therefore, it can be considered a risk-minimising method. Table 7.3 summarises all the data related to the routes covered by the four

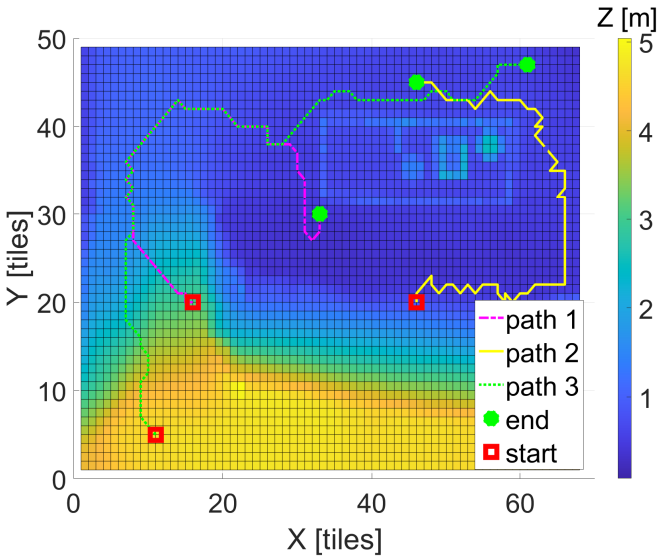


Figure 7.15: Minimum path traversability maps method: optimal paths obtained for the Asguard robot, changing the start and target position.

Table 7.2: Average torques involved for the minimum risk, longer path (MR) and minimum length, larger risk path (ML); / indicates uncompleted paths.

	Average torques $[kg \cdot m^2/s^2]$	
	MR	ML
Asguard	29	34
Coop. Asguard	20	27
5BSLP	83	/
Robotnik	79	/

robots, as the number of tiles travelled and the average risk of the route obtained, where 0 indicates a completely safe path and 1 is an impossible route.

From the analysis, it can be highlighted that two different methods can be considered for selecting the optimal path: the ML which selects the shortest path, and the MR which chooses a longer but the safest cumulative path over the admissible paths. Consequently, two different traversability maps are obtained. Considering Table 7.3 and, in particular, the values related to the MR method, the paths at maximum and minimum risk can be considered Pareto-optimal because there are no other paths that are more traversable (i.e. less risky) and shorter at the same time, based on the particular risk maximum threshold chosen.

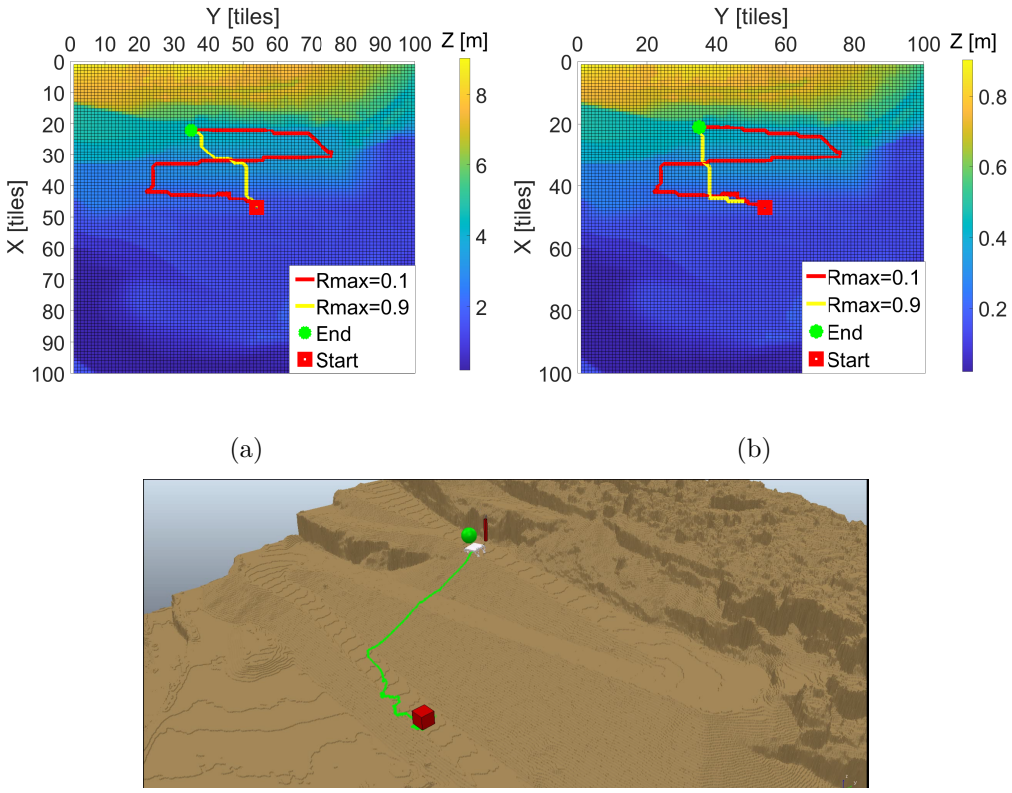


Figure 7.16: Short ($R_{max} = 0.9$) and long ($R_{max} = 0.1$) path related to the 5BSPL robot: (a) paths obtained using the ML, (b) paths obtained using the MR, (c) short path performed in the simulation environment using the MR with $R_{max} = 0.9$.

7.7 Conclusion

The goal of this Chapter was the realization of a simple neural network capable of learning the capabilities of a particular robot, chosen among a series of robots, when it moves through a complex terrain in order to generate a neural model expressing the robot's capabilities. For this goal, a set of high-level features was collected from heightmaps and used to learn neural structures that can classify if a specific area of the map is traversable for the selected robot.

Considering the accuracy values obtained using different neural structures, a shallow network such as a multilayer perceptron that achieved interesting performances, was chosen. The traversability maps generated with MLPs can be used to evaluate the optimal path between the starting and target positions. The concept of Pareto optimality was then introduced to choose the best robotic structure and the right compromise between

Table 7.3: Length of the path expressed in the number of crossed tiles and average risk referred to the paths followed by the four robots. The upper bound for the acceptable risk was considered for the two cases, R_{max}=0.9, and R_{max}=0.1.

		Minimum length (ML)		Minimum risk (MR)	
		$\mathcal{L}(\overline{V}_{47,54\rightarrow 22,35})$	Average risk	$\mathcal{L}(V_{47,54\rightarrow 22,35})$	Average risk
Robotnik	R_{max}=0.9	28	0.58	57	0.12
	R_{max}=0.1	143	0.39	224	0.01
5BSPL	R_{max}=0.9	33	0.35	41	0.14
	R_{max}=0.1	143	0.05	146	0.05
Asguard	R_{max}=0.9	37	0.21	47	0.13
	R_{max}=0.1	134	0.06	153	0.03
Coop. Asguard	R_{max}=0.9	40	0.22	114	0.07
	R_{max}=0.1	134	0.06	155	0.28

the path length and the associated risk.

It was demonstrated that, in certain cases, the possibility of assembling robots leads to an improvement in individual capabilities.

Comparisons with more complex neural networks already used in the literature, such as convolutional neural networks, showed that the proposed shallow networks can provide comparable results in the generation of traversability maps.

Chapter 8

Conclusions and future developments

The results reported in the previous chapters are, of course, encouraging and give the possibility to pursue these research topics related to the study of neuro-inspired robots and applications. In fact, there are still some open points that need a deeper investigation and consideration.

Considering the traversability maps generation problem, the implemented algorithm could be extended to include different types of terrains. In particular, the focus could be on the acquisition of further information, beyond the terrain morphology, regarding the type and consistency (e.g., sand, gravel, clay, rubble, silt, etc.) and creating specialised networks able to model the robot behaviour in each specific terrain condition. The work opens the way to embedded, edge computing-based solutions to implement the optimal route planning directly on board the robot. From a real application point of view, in terrains affected by landslides, the effort lies in deriving an accurate model of the chosen robot.

Considering the real structure of the Mini Cheetah robot, it includes the external PC and the sensor modules that give a neural and a sensing intelligence to the robot. As reported in [22], due to the addition of new hardware components, the actual structure has unavoidably a different weight arrangements, so the dynamics effect modification should be taken into account and duly included in the software framework for the application of the whole methodology developed in [86]. Considering this structure peculiarities, the theoretical results related to the LMPC and NNMPC obtained in [31, 30] should be validated on the real final structure. Furthermore, regarding the LSM methodology applied to the quadruped locomotion shown in [60], several future applications and studies could involve the application of the developed system on the real robots based on the promising results reported.

List of Acronyms

- AIC: Akaike Information Criterion
- ARNN: Artificial Recurrent Neural Networks
- CNN: Cellular Neural Network
- CoM: Center of Mass
- CoT: Cost of Transport
- CPG: Central Patter Generator
- DoF: Degree of Freedom
- ESM: Echo State Machine
- FHN: FitzHugh-Nagumo
- FNN: Feed-forward Neural Network
- GoF: Goodness of Fit
- GPS: Global Positioning System
- GRFs: Ground Reaction Forces
- LMPC: Linear Model Predictive Control
- LSM: Liquid State Machine
- MaxAE: Maximum Absolute Error
- MIMO: Multi-Input Multi-Output
- MPC: Model Predictive Control
- MSE: Mean Squared Error
- NN: Neural Network
- NNMPC: Neural Network Model Predictive Control
- NRMSE: Normalized Root Mean Square Error
- PID: Proportional Integrative Derivative
- PWL: Piece Wise Linear
- RC: Reservoir Computing
- RNN: Recursive Neural Network
- SNN: Spiking Neural Network
- TRQs: Torques

List of Figures

List of Tables

Bibliography

[1] E Arena, P Arena, and L Patanè. Efficient hexapodal locomotion control based on flow-invariant subspaces. *IFAC Proceedings Volumes*, 44(1):13758 – 13763, 2011. 18th IFAC World Congress.

[2] David Zarrouk and Ronald S. Fearing. Cost of locomotion of a dynamic hexapedal robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 2548–2553, 2013.

[3] Christine Huffard, Farnis Boneka, and Robert Full. Underwater bipedal locomotion by octopuses in disguise. *Science (New York, N.Y.)*, 307:1927, 04 2005.

[4] Maceo-Giovanni Angeli, Alessandro Pasuto, and Sandro Silvano. A critical review of landslide monitoring experiences. *Engineering Geology*, 55(3):133–147, 2000.

[5] Luca Patané. Bio-inspired robotic solutions for landslide monitoring. *Energies*, 12(7), 2019.

[6] E. Rohmer, S. P. N. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.

[7] Marc-Oliver Gewaltig and Markus Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2(4):1430, 2007.

[8] Stanford Artificial Intelligence Laboratory et al. Robotic operating system.

[9] Michael Kattan and Thomas Gerds. The index of prediction accuracy: an intuitive measure useful for evaluating risk prediction models. *Diagnostic and Prognostic Research*, 2, 05 2018.

[10] Niyati Gupta. Accuracy, sensitivity and specificity measurement of various classification techniques on healthcare data. *IOSR Journal of Computer Engineering*, 11:70–73, 01 2013.

[11] Ekaterina V. Chimitova and Evgeniia S. Chetvertakova. Goodness-of-fit testing for the degradation models in reliability analysis. In *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, pages 45–48, 2018.

[12] Jie Ding, Vahid Tarokh, and Yuhong Yang. Bridging aic and bic: A new criterion for autoregression. *IEEE Transactions on Information Theory*, 64(6):4024–4043, 2018.

[13] Potiwat Ngamkajornwiwat, Jettanan Homchanthanakul, Pitiwut Teerakittikul, and Poramate Manoonpong. Bio-inspired adaptive locomotion control system for online adaptation of a walking robot on complex terrains. *IEEE Access*, 8:91587–91602, 2020.

[14] César Ferreira and Cristina Santos. A sensory-driven controller for quadruped locomotion. *Biological Cybernetics*, 111, 02 2017.

[15] P. Arena, L. Fortuna, and M. Branciforte. Reaction-diffusion cnn algorithms to generate and control artificial locomotion. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 46(2):253–260, 1999.

[16] P. Arena, S. Baglio, L. Fortuna, and G. Manganaro. Cellular neural networks: A survey. *IFAC Proceedings Volumes*, 28(10):43–48, 1995. 7th IFAC Symposium on Large Scale Systems: Theory and Applications 1995, London, UK, 11-13 July, 1995.

[17] Paolo Arena and Luca Patanè, editors. *Spatial Temporal Patterns for Action-Oriented Perception in Roving Robots II, An Insect Brain Computational Model*, volume 21 of *Cognitive Systems Monographs*. Springer, 2014.

[18] W. Wang and J. Slotine. On partial contraction analysis for coupled nonlinear oscillators. *Biological Cybernetics*, 92:38–53, 2004.

[19] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[20] Paolo Arena, Luca Patanè, and Salvatore Taffara. Energy efficiency of a quadruped robot with neuro-inspired control in complex environments. *Energies*, 14(2), 2021.

[21] Benjamin Katz, Jared Di Carlo, and Sangbae Kim. Mini cheetah: A platform for pushing the limits of dynamic quadruped control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6295–6301, 2019.

[22] Paolo Arena, Fabio Di Pietro, Alessia Li Noce, Salvatore Taffara, and Luca Patanè. Assessment of navigation capabilities of mini cheetah robot for monitoring of landslide terrains. In *2021 IEEE 6th International Forum on Research and Technology for Society and Industry (RTSI)*, pages 540–545, 2021.

[23] Jun Zhao and Yong-Bin Kim. Circuit implementation of fitzhugh-nagumo neuron model using field programmable analog arrays. pages 772 – 775, 09 2007.

[24] Paolo Arena, Luca Patané, and Angelo Giuseppe Spinosa. A nullcline-based control strategy for pwl-shaped oscillators. *Nonlinear Dynamics*, 97, 07 2019.

[25] Arnaud Tonnelier. The mckean's caricature of the fitzhugh–nagumo model i. the space-clamped system. *SIAM Journal on Applied Mathematics*, 63(2):459–484, 2003.

[26] Paolo Arena, Luca Patané, and Angelo Giuseppe Spinosa. A nullcline-based control strategy for pwl-shaped oscillators. *Nonlinear Dynamics*, 97, 07 2019.

[27] P. Arena, A. Bonanzinga, and L. Patané. Role of feedback and local coupling in cnns for locomotion control of a quadruped robot. In *CNNA 2018; The 16th International Workshop on Cellular Nanoscale Networks and their Applications*, pages 1–4, 2018.

[28] Paolo Arena, Andrea Bonanzinga, and Luca Patané. *From Parallel to Emergent Computing*, pages 547–574. 03 2019.

[29] Holk Cruse, Thomas Kindermann, Michael Schumm, Jeffrey Dean, and Josef Schmitz. Walknet—a biologically inspired network to control six-legged walking. *Neural Networks*, 11(7):1435–1447, 1998.

[30] Paolo Arena, Luca Patanè, Pierfrancesco Sueri, and Salvatore Taffara. A data-driven neural network model predictive steering controller for a bio-inspired quadruped robot. *IFAC-PapersOnLine*, 54(17):93–98, 2021. 6th IFAC Conference on Analysis and Control of Chaotic Systems CHAOS 2021.

[31] Paolo Arena, Pierfrancesco Sueri, Salvatore Taffara, and Luca Patanè. Mpc-based control strategy of a neuro-inspired quadruped robot. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[32] Kiyotoshi Matsuoka. Mechanisms of frequency and pattern control in the neural rhythm generators. *Biological cybernetics*, 56:345–53, 02 1987.

[33] Yasuhiro Fukuoka, Yasushi Habu, and Takahiro Fukui. A simple rule for quadrupedal gait generation determined by leg loading feedback: A modeling study. *Scientific reports*, 5:8169, 02 2015.

[34] K.J. Astrom and T. Hagglund. Advanced PID control. *Research Triangle Park, NC: Instrumentation, Systems, and Automation Society*, 2006.

[35] Tao Sun, Xiaofeng Xiong, Zhendong Dai, and Poramate Manoonpong. Small-sized reconfigurable quadruped robot with multiple sensory feedback for studying adaptive and versatile behaviors. *Frontiers in Neurorobotics*, 14, 02 2020.

[36] Frank Pasemann, Manfred Hild, and Keyan Ghazi-Zahedi. So(2)-networks as neural oscillators. pages 144–151, 06 2003.

[37] Long Zhou, Li Xie, and Xiao-Jun Tong. Recursive neural networks and its application in forecasting the state of electric power equipment. In *2007 International Conference on Machine Learning and Cybernetics*, volume 5, pages 2801–2804, 2007.

[38] Alejandro Chinea. Understanding the principles of recursive neural networks: A generative approach to tackle model complexity. pages 952–963, 2009.

[39] Zhiyuan Tang, Dong Wang, and Zhiyong Zhang. Recurrent neural network training with dark knowledge transfer. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904, 2016.

[40] Zhang Yifan, Qian Fengchen, and Xiao Fei. Gs-rnn: A novel rnn optimization method based on vanishing gradient mitigation for hrrp sequence estimation and recognition. In *2020 IEEE 3rd International Conference on Electronics Technology (ICET)*, pages 840–844, 2020.

[41] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5457–5466, 2018.

[42] Wolfgang Maass and Henry Markram. On the computational power of circuits of spiking neurons. *Journal of Computer and System Sciences*, 69(4):593–616, 2004.

[43] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, 2019.

[44] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007. Echo State Networks and Liquid State Machines.

[45] Nicholas Soures and Dhireesha Kudithipudi. Spiking reservoir networks: Brain-inspired recurrent algorithms that use random, fixed synaptic strengths. *IEEE Signal Processing Magazine*, 36(6):78–87, 2019.

[46] Nicholas Soures and Dhireesha Kudithipudi. Deep liquid state machines with neural plasticity for video activity recognition. *Frontiers in Neuroscience*, 13, 2019.

[47] Tadashi Yamazaki and Shigeru Tanaka. The cerebellum as a liquid state machine. *Neural networks : the official journal of the International Neural Network Society*, 20:290–7, 05 2007.

[48] Andreea Lazar, Gordon Pipa, and Jochen Triesch. Fading memory and time series prediction in recurrent networks with different forms of plasticity. *Neural networks : the official journal of the International Neural Network Society*, 20:312–22, 05 2007.

[49] Alberto Patiño-Saucedo, Horacio Rostro-González, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. Liquid state machine on spinnaker for spatio-temporal classification tasks. *Frontiers in Neuroscience*, 16, 2022.

[50] Herbert Jaeger, Mantas Lukoševičius, Dan Popovici, and Udo Siewert. Optimization and applications of echo state networks with leaky- integrator neurons. *Neural Networks*, 20(3):335–352, 2007. Echo State Networks and Liquid State Machines.

[51] Jochen J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007. Echo State Networks and Liquid State Machines.

[52] Yanbo Xue, Le Yang, and Simon Haykin. Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20(3):365–376, 2007. Echo State Networks and Liquid State Machines.

[53] Ganesh Venayagamoorthy. Online design of an echo state network based wide area monitor for a multimachine power system. *Neural networks : the official journal of the International Neural Network Society*, 20:404–13, 05 2007.

[54] M.D. Skowronski and John Harris. Minimum mean squared error time series classification using an echo state network prediction model. pages 4 pp. – 3156, 06 2006.

[55] Matthew H. Tong, Adam D. Bickett, Eric M. Christiansen, and Garrison W. Cottrell. Learning grammatical structure with echo state networks. *Neural Networks*, 20(3):424–432, 2007. Echo State Networks and Liquid State Machines.

[56] Alberto Patiño-Saucedo, Horacio Rostro-González, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. Liquid state machine on spinnaker for spatio-temporal classification tasks. *Frontiers in Neuroscience*, 16, 2022.

[57] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. The heidelberg spiking data sets for the systematic evaluation of spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2020.

[58] Gideon Gbenga Oladipupo. Research on the concept of liquid state machine, 2019.

[59] Wolfgang Maass. *Motivation, theory, and applications of liquid state machines*, pages 275–296. Imperial College Press, 1 edition, 2011.

[60] Paolo Arena, Cusimano Maria Francesca Pia, Patanè Luca, Emmanuele Lorenzo, Manoonpong Poramate, and Taffara Salvatore. Ground reaction force estimation in a quadruped robot via liquid state networks. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[61] Paolo Arena, Alessia Li Noce, Luca Patanè, and Salvatore Taffara. Insect-inspired spiking neural controllers for adaptive behaviors in bio-robots. *IEEE Instrumentation Measurement Magazine*, 25(9):19–27, 2022.

[62] E.M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003.

[63] tMisha Tsodyks, Asher Uziel, and Henry Markram. Synchrony generation in recurrent networks with frequency-dependent synapses. *Journal of Neuroscience*, 20(1):RC50–RC50, 2000.

[64] Jacques Kaiser, Rainer Stal, Anand Subramoney, Arne Roennau, and Rüdiger Dillmann. Scaling up liquid state machines to predict over address events from dynamic vision sensors. *Bioinspiration and Biomimetics*, 12, 06 2017.

[65] Mario Calandra, Luca Patanè, Tao Sun, Paolo Arena, and Poramate Manoonpong. Echo state networks for estimating exteroceptive conditions from proprioceptive states in quadruped robots. *Frontiers in Neurorobotics*, 15, 2021.

[66] Jingtao Lei, Feng Wang, Huangying Yu, Tianmiao Wang, and Peijiang Yuan. Energy efficiency analysis of quadruped robot with trot gait and combined cycloid foot trajectory. *Chinese Journal of Mechanical Engineering*, 27:138–145, 01 2014.

[67] Christopher Y. Brown, Dana E. Vogtmann, and Sarah Bergbreiter. Efficiency and effectiveness analysis of a new direct drive miniature quadruped robot. In *2013 IEEE International Conference on Robotics and Automation*, pages 5631–5637, 2013.

[68] Sangbae Kim and Patrick Wensing. Design of dynamic legged robots. *Foundations and Trends in Robotics*, 5:117–190, 01 2017.

[69] Dario Guastella, Luciano Cantelli, Carmelo Melita, and Giovanni Muscato. A global path planning strategy for a ugv from aerial elevation maps for disaster response. pages 335–342, 01 2017.

[70] Peng Zhang. Chapter 2 - industrial control engineering. In Peng Zhang, editor, *Advanced Industrial Control Technology*, pages 41–70. William Andrew Publishing, Oxford, 2010.

[71] J.B. Rawlings. Tutorial overview of model predictive control. *IEEE Control Systems Magazine*, 20(3):38–52, 2000.

[72] James B. Rawlings. Tutorial: model predictive control technology. *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, 1:662–676 vol.1, 1999.

[73] Jan Magnus. Static optimization. pages 129–159, 05 2019.

[74] Don Soloway and P.J. Haley. Neural generalized predictive control. *IEEE International Symposium on Intelligent Control*, pages 277 – 282, 10 1996.

[75] Mathworks. Srchbac, 1-D minimization using backtracking. https://it.mathworks.com/help/deeplearning/ref/srchbac.html. Accessed: 2022-11-15.

[76] P. Arena and L. Patanè. Contributions of cnn to bio-robotics and brain science. In Andrew Adamatzky and Guanrong Chen, editors, *Chaos, CNN, Memristors and Beyond*, pages 56–82. World Scientific, 2013.

[77] Paolo Arena, S. Castorina, Luigi Fortuna, Mattia Frasca, and M. Ruta. A cnn-based chip for robot locomotion control. volume 52, pages III–510, 06 2003.

[78] M.-A. Belabbas and J.-J. E. Slotine. Factorizations and partial contraction of nonlinear systems. In *Proceedings of the 2010 American Control Conference*, pages 3440–3445, 2010.

[79] J. E. Dennis and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1996.

[80] Michele Focchi, Victor Barasuol, Marco Frigerio, Darwin G. Caldwell, and Claudio Semini. *Slip Detection and Recovery for Quadruped Robots*, pages 185–199. Springer International Publishing, Cham, 2018.

[81] Tamás Haidegger, Levente Kovács, Radu-Emil Precup, Stefan Preitl, Balázs Benyó, and Zoltán Benyó. Cascade control for telerobotic systems serving space medicine*. *IFAC Proceedings Volumes*, 44(1):3759–3764, 2011. 18th IFAC World Congress.

[82] Jihee Han. An efficient approach to 3d path planning. *Information Sciences*, 478:318–330, 2019.

[83] Rezia M. Molfino, Roberto P. Razzoli, and Matteo Zoppi. Autonomous drilling robot for landslide monitoring and consolidation. *Automation in Construction*, 17(2):111–121, 2008. 22nd Symposium on Automation and Robotics in Construction, ISARC 2005.

[84] Klaus Schilling and Christoph Jungius. Mobile robots for planetary exploration. *IFAC Proceedings Volumes*, 28(11):109–119, 1995. 2nd IFAC Conference on Intelligent Autonomous Vehicles 1995, Espoo, Finland, 12-14 June 1995.

[85] Ioannis Kontolatis, Dimitris Myrisiotis, Iosif Paraskevas, Evangelos Papadopoulos, Guido Croon, and Dario Izzo. Quadruped optimum gaits analysis for planetary exploration. 05 2013.

[86] Paolo Arena, Luca Patanè, and Salvatore Taffara. Learning risk-mediated traversability maps in unstructured terrains navigation through robot-oriented models. *Information Sciences*, 576:1–23, 2021.

[87] Paolo Arena, Carmelo Fabrizio Blanco, Alessia Li Noce, Salvatore Taffara, and Luca Patanè. Learning traversability map of different robotic platforms for unstructured terrains path planning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[88] Flora Amato, Nicola Mazzocca, Francesco Moscato, and Emilio Vivenzio. Multilayer perceptron: An intelligent model for classification and intrusion detection. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 686–691, 2017.

[89] Anthony J. Myles, Robert N. Feudale, Yang Liu, Nathaniel A. Woody, and Steven D. Brown. An introduction to decision tree modeling. *Journal of Chemometrics*, 18(6):275–285, 2004.

[90] Koksal Karadas, Muhammad Tariq, Mohammad Tariq, and Ecevit Eyduran. Measuring predictive performance of data mining and artificial neural network algorithms for predicting lactation milk yield in indigenous akkaraman sheep. *Pakistan Journal of Zoology*, 49:1–7, 09 2016.

[91] Soham Pathak, Indivar Mishra, and Aleena Swetapadma. An assessment of decision tree based classification and regression algorithms. In *2018 3rd International Conference on Inventive Computation Technologies (ICICT)*, pages 92–95, 2018.

[92] Leo Breiman. Random forests. *Machine Learning*, 45, 2001.

[93] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

[94] Jayati Vaish, Stuti Shukla Datta, and k Seethalekshmi. Short-term load forecasting using bootstrap aggregation based ensemble method. In *2021 7th International Conference on Electrical Energy Systems (ICEES)*, pages 245–249, 2021.

[95] Ahlem Bougarradh, Slim M'hiri, and Faouzi Ghorbel. Introduction of the bootstrap resampling in the generalized mixture estimation. In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–6, 2008.

[96] Ariel Felner. Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm. In *SOCS*, 2011.

[97] Summit-xl mobile robot. Accessed: 2020-12-25.

[98] G. Kenneally, A. De, and D. E. Koditschek. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1(2):900–907, 2016.

[99] M. Eich, F. Grimminger, and F. Kirchner. A versatile stair-climbing robot for search and rescue applications. In *2008 IEEE International Workshop on Safety, Security and Rescue Robotics*, pages 35–40, 2008.

[100] E. Arena, P. Arena, and L. Patané. Cpg-based locomotion generation in a drosophila inspired legged robot. In *2012 4th IEEE RAS EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 1341–1346, June 2012.

[101] Dario Guastella, Luciano Cantelli, Carmelo Melita, and Giovanni Muscato. A global path planning strategy for a ugv from aerial elevation maps for disaster response. pages 335–342, 01 2017.

[102] Panagiotis Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373 – 1385, 2013.

[103] Ali Jahan and Kevin Edwards. *Multi-Criteria Decision Analysis for Supporting the Selection of Engineering Materials in Product Design*. 01 2013.

[104] S. Choudhury, C. M. Dellin, and S. S. Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3742–3749, 2016.

[105] Alexander Lavin. A pareto front-based multiobjective path planning algorithm. *Artificial Intelligence*, 2015.