Sant'Anna
School of Advanced Studies – Pisa

DOCTORAL THESIS

Brain-inspired methods for adaptive and predictive control of humanoid robots

Author:
Lorenzo VANNUCCI

Supervisor:
Prof. Cecilia Laschi

Tutor:
Dr. Egidio Falotico

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Ph.D. Programme in BioRobotics

One might say we ought to make the human better.
That is impossible, so we will make the robot more foolproof.

—Isaac Asimov, *The Naked Sun*

Abstract

This thesis advances the field of control of humanoid robots, focusing on brain-inspired approaches to the problem.

Human beings are able to negotiate complex environments thanks to their advanced bodies and brains. The human body is a complex machine that is able to perceive the world through different sensors modalities and to perform fine complex motions with a high number of strong, redundant, compliant actuators. However, humans would not be able to perform such feats if it were not for the orchestration performed by the nervous system, and especially by the brain. The human brain is the most advanced of the animal kingdom and is capable of displaying cognitive capabilities as well as motor control strategies. In particular, the main feature of the human brain is being capable to process the massive amount of sensory information received and to use it to perform both cognitive and motor tasks, often at the same time. Concerning, motion control, the brain is able not only to generate descending motor signal to the muscles that perform coordinated fine movements, but also to combine proprioceptive and exteroceptive sensory information to adapt movements depending on different conditions, in a predictive and anticipative fashion. Predicting changes in the environment and reacting accordingly is a must-have feature for robots that need to interact with an unstructured and dynamic environment that has been built by humans for themselves. And such is the case for humanoid robots, that are supposed to be employed either as service assistants and in search and rescue scenarios.

This works focus on control strategies that take inspiration from the human brain and that show some of these predictive and adaptive capabilities. This has been achieved by employing *neuro-controllers*, controllers that embed algorithms mimicking the role of brain areas, specifically related to the task at hand. In particular, two technologies have been considered to implement neuro-controllers: machine learning approaches and Spiking Neural Networks (SNN). The formers have been widely used in different contexts, from artificial intelligence to data mining, and for many years in robotics to complement classic control approaches. In the context of this work, two different controllers have been developed that make use of machine learning methods: a predictive visual tracking architecture and a gaze stabilization controller provided with sensory-motor anticipation. Conversely, SNN have been mostly used in neuroscientific context to perform detailed simulations of the human brain. Albeit their use in robotics have been limited, there is a rising interest in this kind of neural simulations that can take the bioininspiration of robotic controllers to the next level. Due to research in neuro-controllers with SNN being in its infancy, the robotics community lacked effective tools to develop them. As part of this work, in the framework of the Human Brain Project, a contribution to the development of a comprehensive tool for the modelling and testing of neuro-controllers embedding SNN (the Neurorobotics Plaform) has been given. Moreover, this tool has been employed to show the effectiveness of the approach with simple examples that include a basic neuro-controller for visual tracking and controller relying on a retinal model for visual perception. This thesis also shows preliminary results toward the implementation of complex control architectures with SNN, focusing on developing low level translation mechanism for actuators and sensors based on simulation of the spinal cord circuitry.

List of publications

In the framework of the present thesis, the following contributions were published.

Journal papers

Falotico, E., L. Vannucci, A. Ambrosano, U. Albanese, S. Ulbrich, J. C. Vasquez Tieck, G. Hinkel, J. Kaiser, I. Peric, O. Denninger, N. Cauli, M. Kirtay, A. Ronneau, G. Klinker, A. von Arnim, L. Guyot, D. Peppicelli, P. Martinez-Cañada, E. Ros, P. Maier, S. Weber, M. Huber, D. Plecher, F. Röhrbein, S. Deser, A. Roitberg, P. van der Smagt, R. Dillmann, P. Levi, C. Laschi, A. Knoll, and M.-O. Gewaltig (2017). "Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform". In: *Frontiers in neurorobotics* 11, p. 2.

Hinkel, G., H. Groenda, S. Krach, L. Vannucci, O. Denninger, N. Cauli, S. Ulbrich, A. Roennau, E. Falotico, M.-O. Gewaltig, et al. (2017). "A framework for coupled simulations of robots and spiking neuronal networks". In: *Journal of Intelligent & Robotic Systems* 85.1, pp. 71–91.

Vannucci, L., E. Falotico, and C. Laschi (2017). "Proprioceptive Feedback through a Neuromorphic Muscle Spindle Model". In: *Frontiers in Neuroscience* 11, p. 341.

Vannucci, L., E. Falotico, S. Tolu, V. Cacucciolo, P. Dario, H. H. Lund, and C. Laschi (2017). "A comprehensive gaze stabilization controller based on cerebellar internal models". In: *Bioinspiration & biomimetics* 12.6, p. 065001.

Conference proceedings

Ambrosano, A., L. Vannucci, U. Albanese, M. Kirtay, E. Falotico, P. Martínez-Cañada, G. Hinkel, J. Kaiser, S. Ulbrich, P. Levi, C. Morillas, A. Knoll, M.-O. Gewaltig, and C. Laschi (2016). "Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9793, pp. 16–27.

Vannucci, L., E. Falotico, S. Tolu, P. Dario, H. Lund, and C. Laschi (2016). "Eye-head stabilization mechanism for a humanoid robot tested on human inertial data". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9793, pp. 341–352.

Vannucci, L., S. Tolu, E. Falotico, P. Dario, H. Lund, and C. Laschi (2016). "Adaptive gaze stabilization through cerebellar internal models in a humanoid robot". In: *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*. Vol. 2016-July, pp. 25–30.

Falotico, E., L. Vannucci, N. Di Lecce, P. Dario, and C. Laschi (2015). "A bio-inspired model of visual pursuit combining feedback and predictive control for a humanoid robot". In: *Proceedings of the 17th International Conference on Advanced Robotics, ICAR 2015*, pp. 188–193.

Hinkel, G., H. Groenda, L. Vannucci, O. Denninger, N. Cauli, and S. Ulbrich (2015). "A domain-specific language (DSL) for integrating neuronal networks in robot control". In: *ACM International Conference Proceeding Series*. Vol. 21-July-2015, pp. 9–15.

Vannucci, L., A. Ambrosano, N. Cauli, U. Albanese, E. Falotico, S. Ulbrich, L. Pfotzer, G. Hinkel, O. Denninger, D. Peppicelli, L. Guyot, A. Von Arnim, S. Deser, P. Maier, R. Dillman, G. Klinker, P. Levi, A. Knoll, M.-O. Gewaltig, and C. Laschi (2015). "A visual tracking model implemented on the iCub robot as a use case for a novel neurorobotic toolkit integrating brain and physics simulation". In: *IEEE-RAS International Conference on Humanoid Robots*. Vol. 2015-December, pp. 1179–1184.

Vannucci, L., E. Falotico, N. Di Lecce, P. Dario, and C. Laschi (2015). "Integrating feedback and predictive control in a Bio-inspired model of visual pursuit implemented on a humanoid robot". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9222, pp. 256–267.

Contents

Chapter 1

Introduction

This thesis is a contribution to the field of control of humanoid robots. Driven by bio-inspiration, it explores the use of different neuroscientific concepts to create novel control strategies for different tasks.

1.1 Challenges of humanoid robotics

The field of humanoid robotics is an important branch of biomimetic robotics which ultimate goal is to create machines that not only resemble human beings but that are also intelligent and able to act, reason and interact like and possibly with humans (Fukuda, Dario, and Yang, 2017). Such robots would be extremely versatile and could quickly permeate a society that has already accepted the presence of specialized service robots, such as the ones for vacuum cleaning (J. L. Jones, 2006) and mowing (Prassler et al., 2016), as well as self-driving cars (Broggi et al., 2016). In particular, humanoid robots could in principle take advantage of their morphology to handle objects that are made to fit human hands or to navigate environments that have been built by humans for themselves (Brooks et al., 2004). Therefore, it easy to see how humanoid robots can assist or even replace humans in so-called 3D-jobs (dirty, dangerous and demanding), with potential applications like service in domestic environments, emergency operation in hazardous areas or spatial exploration (Fitzpatrick et al., 2016).

However, humanoid robots have yet to prove really effective in unstructured environments. In 2015, the finals of the DARPA Robotics Challenge (DRC) saw 23 teams competing to have their humanoid robot perform the different tasks in a simulated disaster response scenario. The competition was motivated by the accident of the Fukushima Daiichi nuclear reactor, hit by a tsunami in 2001. The team had to teleoperate the robots, through degraded communication channels, in order to perform diverse tasks such as driving a car, turning a valve, drilling a hole in the wall with a tool, walking over uneven terrain or climbing steps (Krotkov et al., 2017). In practice, the DRC provided a benchmark of the capabilities of state-of-the-art humanoid robots in unstructured environments. The results of this challenge highlighted that there is still a long way to go before humanoids can reliably and autonomously operate in such conditions. In particular the most evident problems of the participating robots were the slowness and the non-capability of recovery after faults. One common cause for the two issues is the lack of autonomy and adaptivity to unexpected situations. In fact, it appears that, while the motion execution itself was performed slowly primarily to avoid falling, some robots were inactive during 74% of the time, waiting for the operators to gain situational awareness and carefully plan the course of action (Fallon et al., 2015). This was because even the smallest changes in the area layout or tiny errors in motion planning led to catastrophic failures (Guizzo and Ackerman, 2015).

Clearly, there is a need for robots that, even if not capable of deciding the action to take in complete autonomy, have some degree of adaptation in performing the task at hand so that unexpected changes in the conditions do not result in failures. Given that a non-athletic human could complete all tasks 20 times faster than the best competing robot, there are more aspects of the human body and behaviour that were neglected when designing the competing robots. From the hardware point of view, most of the robots employed completely rigid joints with few exceptions such as the ATLAS, the WALKMAN (Tsagarakis et al., 2016) and the Hydra robots (Kaminaga et al., 2010). A more compliant joint structure, albeit more complex to control, would enhance the robots with some hardware adaptation to small, unexpected forces or slightly different foot contact points. From the control side, the robots should be able to anticipate both external situations as well as proprioceptive sensory information in order to quickly adapt their behaviour without the need for operator intervention. As the human brain is particularly well versed in predicting and adapting, it is natural to consider beneficial the inclusion of brain-inspired mechanisms, that originated from neuroscientific findings, into robotic control loops.

1.2 Neuroscience and robotics

As we get more insights on the inner working of biological system, these become a fruitful source of inspiration for roboticists (Iida and A. J. Ijspeert, 2016). In particular, in the past years there have been many synergies between the fields of neuroscience and robotics, beneficial for both sides. This is especially true for humanoid robots that have served as physical embodiments for neuroscientific theories emerging from studies of the human brain. In fact, there were humanoid robotic platform that were built to support research in different aspects of cognition such as the iCub for studies on cognition (Metta, Sandini, et al., 2008) and the SABIAN for studies on human-like motions (Muscolo et al., 2012).

Conversely, many control architectures inspired from brain functionalities have been employed for various tasks, with different degrees of bio-inspiration and biomimetism, giving birth to the so-called *neuro-controllers*. One of the early examples of this kind of controllers was an artificial neural network with a brain-like hierarchical architecture for the control of the trajectory of a robotic manipulator (Miyamoto et al., 1988). In (Reeke, Sporns, and Edelman, 1990), a learning mechanism based on the theory of neuronal group selection enabled the Darwin III robot to perform different behaviours such as pattern recognition and sensorimotor coordination. Cox and colleagues have employed a mechanism similar to the neuromodulation of the mammalian brain to implement behaviours such as attention and action selection (Cox and Krichmar, 2009). More complete brain-inspired models have also been used as neuro-controllers such as the rat hippocampus (Burgess, Donnett, and O'Keefe, 1998) or rat basal ganglia (Fox et al., 2009).

In parallel, advancements in computational neuroscience led to the development of tools for detailed simulations of spiking neural networks (SNNs), that, differently from artificial neural networks, consist of unit that mimic the neural dynamics and the activity of synapses. Although such networks are more computationally expensive to simulate, there is the benefit of being able to tap into vast model literature delivered by neuroscientific research. This, combined with birth of hardware platforms dedicated to the simulation of such networks (neuromorphic hardware), has led to a new wave of neuro-controllers that include more realistic models of the human nervous system, simulated through spiking neural networks (Knoll and Gewaltig,

2016). Although the research can still be considered in its infancy, few examples can be already found in literature. For instance, in (Lund, Webb, and Hallam, 1998) a spiking model of cricket phonotaxis was evaluated on a mobile robot, while a spiking neural network with spike timing dependant plasticity was employed to learn to control of a 4 degrees of freedom robotic arm (Bouganis and Shanahan, 2010). Clearly, spiking neural network are more well suited to be coupled with musculoskeletal systems (Sreenivasa, Ayusawa, and Nakamura, 2016) or muscle-like actuation mechanisms (Richter et al., 2016).

1.3 Models of prediction and adaptation

As stated in Section 1.1, many robots lack predictive and adaptive capabilities that are essential to negotiate unstructured environments. In fact, human beings are unconsciously able to predict changes in their sensory systems (Berthoz, 2002), by using internal models that provide representations of their own bodies as well as external object dynamics (Johansson, 1998; Miall and Wolpert, 1996; Nguyen-Tuong and Peters, 2011). Three types of internal models emerged by neuroscientific findings (Miall and Wolpert, 1996): *forward models*, that are able to predict future sensory information from past perceptions and planned actions; *environment models* predict the dynamics of external objects or agents; *inverse models* provide the actions needed to reach a desired state, given the actual one. There is evidence that the brain area responsible for learning and storing internal models is the cerebellum (Wolpert, Miall, and Kawato, 1998). In humans, internal models do not only provide inputs to the brain areas involved in motion generation, but they effectively work in parallel with them. In fact, the primary motor cortex is incorporated into the cerebrocerebellar loops that connect it with recurrent connection with the cerebellar hemisphere (Ito, 2012).

In robotics, predictors similar to these internal models have been used to implement anticipatory sensory motor systems. For instance, in (Gross et al., 1999) and (Hoffmann, 2007) forward models were used to predict visual information on mobile robotic platforms. On the other hand environment models were for instance employed to catch flying objects (Bäuml et al., 2011; Kim and Billard, 2012; Kober, Glisson, and Mistry, 2012). In all these works, internal models are only used to perform anticipated actions, maintaining a strict sequentiality between (anticipated) perception and the planning of the action. These architectures have an intrinsic flaw that should be addressed by the system: if the internal model is not accurate, the prediction cannot be trusted. Inaccuracy of forward and inverse internal models can occur for various reason including unexpected loads or faults. In such cases, the model must be able to dynamically adapt to the new conditions, or the control will be inaccurate. In the case of environmental models, inaccuracies could occur because the external object dynamics is not predictable. In this case, these simple architectures will not work.

A more advanced technique technique that exploit forward models is *model predictive control* (MPC), in which a forward model is used to compute future states of the system, at every control step, with the goal of optimizing the control signals with respect to some constraints (Camacho and Alba, 2013). This kind of controller has been successfully employed in conjunction with the zero moment point technique for biped walking (Wieber, 2006; Diedam et al., 2008), to perform whole-body control (Koenemann et al., 2015) or balancing tasks (Henze, Ott, and Roa, 2014). Due to the fact that the prediction is dynamically adjusted by being recomputed online at

every step, these models can deal with unexpected sensory inputs to some extent. However, to deal with inaccuracies of the internal model, more robust techniques like min-max MPC (Scokaert and Mayne, 1998) or multi-stage MPC (Lucia, Finkler, and Engell, 2013). These often involve the introduction of uncertainties in the optimization procedures which can lead to exponential growth of the problem and, consequently, higher computational costs.

An alternative approach, generally more computationally efficient, can be found in control systems based on the principle of the *expected perception* (EP). In this architecture, predictions are compared with actual sensory data to get a measure of the accuracy of the prediction. If the prediction is accurate there is no need for corrective actions, while if there is a mismatch the sensory feedback will be used to to compute the corrections and the new predictions (Datteri et al., 2003). EP-based controllers have been implemented for different applications. For instance it has been used to enhance a robotic manipulator with the ability to grasp an object by predicting the tactile image that would be perceived (Laschi, Asuni, Teti, et al., 2006; Laschi, Asuni, Guglielmelli, et al., 2008). In (Moutinho et al., 2011) EP has been used to locate unexpected objects in a cluttered scene by comparing the predicted camera image with the actual one. Finally in (Cauli et al., 2016), an EP-based architecture was developed for reaching moving targets.

EP-based architecture clearly show a certain degree of adaptation due to the corrective actions that can be taken if the prediction is inaccurate. But again, if the internal model becomes inaccurate for any reason, there is no way for the system to use the prediction. Therefore, architectures such as the EP or MPC should be used in conjunction with internal models capable of online adaptation to new circumstances. While several generic implementations of adaptive internal models exist (Tolu, Vanegas, Luque, et al., 2012; Luque et al., 2011), they have almost exclusively been used to implement simple tasks and were never embedded in more complex systems controlling multiple degrees of freedom in conjunction with EP-like mechanisms. Moreover, none of the discussed models has been proven to work in case of drastic changes in the predicted dynamics by using feedback to adjust the internal model and continue again with predictive behaviour. To provide a concrete advancement over the state of the art, the developed models will be specifically developed to address to perform such feats.

1.4 Goals

The main goal of this thesis is to make steps towards addressing the problems described previously such as the lack of adaptiveness when dealing with humanoid robots in unstructured environments. In particular the focus is on the prediction of both external circumstances and of proprioceptive feedback and how this predictions can be used to adjust the behaviour of the robot, resulting in adaptive controllers.

Given that prediction and adaptability are mechanism observable in human behaviour and thus generated by the human brain, and given the effectiveness of using brain-inspired mechanisms to develop control strategies, it is only natural to have resorted to the use of neuro-controllers for developing the models presented in this thesis. In particular, both artificial neural networks and spiking neural networks have been employed.

Finally, by using adaptive internal models implemented through neural methods or learning mechanisms in general, in conjunction with EP-like adjustment mechanisms, the models developed will be able to deal with drastic changes in the predicted internal or external states or unpredictable dynamics, by recovering the predictive behaviour when possible.

In a first phase the research focused on two tasks that make use of prediction to produce adaptive behaviours and that together contribute to improve the visual capabilities of robots: smooth pursuit and gaze stabilization. The former is the act of following a moving target with foveal vision and prediction of the target dynamics must be employed in order to follow the target with no lag. Humans are capable of following both predictable and unpredictable targets, thus a model that employ prediction as well as a switching mechanism to distinguish between the two is presented in Chapter 2. Gaze stabilization, the reflexive act of maintaining the image stable on the retina by compensating for external motions is an essential feature for robots capable of locomotion. As opposed to the smooth pursuit, it relies on the prediction of sensory feedback (also called sensory-motor anticipation) provided by cerebellar internal models to fully cancel the disturbances. A comprehensive model of gaze stabilization that include all the contributing reflexes is presented in Chapter 3.

While the usage of artificial neural network in robotics is a well established technology and was thus used to develop complete complex controllers, there is no definitive answer on how to couple a spiking neural simulation with a robotic platform. For this reason, before trying to develop complex controllers with detailed brain models, it was necessary to develop both software tools and low-level mechanisms for this coupling. In particular, in the framework of the Human Brain Project Subproject 10 - Neurorobotics[1], a contribution was given to the development of simulation platform that couples physical and neural simulations, the Neurorobotics Platform. A description of this piece of software and the preliminary results obtained with it will be presented in Chapter 4. The main conceptual issue when trying to connect robots with SNNs is how to translate analogue information from sensors into electric signals such as spikes and, vice versa, how to convert neural activity into actuation signals. While in many cases researches relies on tailor-made conversions, in Chapter 5, a possible general approach, inspired by the mammalian spinal cord, is presented.

[1] www.humanbrainproject.eu

Chapter 2

Prediction of and adaptation to external motions: smooth pursuit

As a fist example of a task where prediction and adaptation play an important role, visual tracking of a moving target was considered. This task not only involve prediction of the target motion, in order to follow it without delay, but also an adaptation to changes in target dynamics.

2.1 Introduction

Following a moving target with a foveal vision is one of the essential tasks of humans and humanoid robots. Humans accomplish this task through a combination of two forms of eye movements: saccade and smooth pursuit. Saccades are high velocity gaze shifts that bring the image of an object of interest onto fovea. The purpose of smooth pursuit eye movements is to minimize the retinal slip, i.e. the target velocity projected onto the retina, stabilizing the image of the moving object on the fovea. This cannot be achieved by a simple visual negative feedback controller due to the long delays (around 100 ms in the human brain), most of which are caused by visual information processing. During maintained smooth pursuit, the lag in eye movement can be reduced or even cancelled if the target trajectory can be predicted (Fukushima et al., 1990; Wells and G. R. Barnes, 1998; Whittaker and Eaholtz, 1982). The first model of the pursuit system, consisting in a very simple model of the fixations system, was built by Dallos and Jones (Dallos and R. Jones, 1963). This model was designed to demonstrate the importance of including delays in the control of the eye movement during pursuit. One of the first models of pursuit systems was built by Robinson and colleagues (D. A. Robinson, J. Gordon, and S. Gordon, 1986) who designed a model of the pursuit including three feedback loops. The outer feedback loop computed a velocity error (difference between the target velocity and the eye velocity) and the two other feedback loops (a positive one and a negative one) ensure that the model could reproduce adequately the ringing settling to a steady state. If the model of Robinson and colleagues could simulate the pursuit behaviour when the target has a constant velocity, it does not include a prediction term that would allow it to simulate pursuit of periodic target. In a complete study of the prediction

This chapter has been adapted from the articles:

Vannucci, L., E. Falotico, N. Di Lecce, P. Dario, and C. Laschi (2015). "Integrating feedback and predictive control in a Bio-inspired model of visual pursuit implemented on a humanoid robot". In: *Lecture Notes in Computer Science*. Vol. 9222, pp. 256–267.

Falotico, E., L. Vannucci, N. Di Lecce, P. Dario, and C. Laschi (2015). "A bio-inspired model of visual pursuit combining feedback and predictive control for a humanoid robot". In: *Proceedings of the 17th International Conference on Advanced Robotics, ICAR 2015*, pp. 188–193.

mechanisms in human smooth pursuit, Barnes and Asselman (G. Barnes and Asselman, 1991) proposed a model including a predictive component and could also simulate, to a certain extent, head-unrestrained pursuit. Later, Krauzlis and Lisberger (Krauzlis and Lisberger, 1994) prosed a model conposed of three pathways that simulate pursuit behaviour when tracking targets with constant velocity. An important biologically plausible smooth pursuit controller has been proposed by Shibata and Schaal (Shibata, Tabata, et al., 2005). This controller learns to predict the visual target velocity in head coordinates, based on on-line learning of the target dynamics. The models proposed by Shibata and Schaal and improved in (Zambrano et al., 2010) results in a good solution to represent predictive behaviour in the smooth pursuit eye movement.
Although the presented systems may consistently represent specific features of the smooth pursuit eye movement, they do not consider to integrate the major characteristics of visually guided and predictive control of the smooth pursuit. The first instance of a model integrating these aspects of the pursuit dynamics in a coherent model has been proposed by Orban de Xivry and colleagues (Xivry et al., 2013). They proposed that the brain circuitry generating the smooth pursuit eye movement implements the functional equivalent of two Kalman filters (for feedback and predictive control) whose output are optimally combined. Their model aims at reproducing the eye dynamics during the switching between feedback and predicting control and it has a memory based on previous values of the target velocity. The eye controller, designed specifically for the human eye and the memory approach makes this model not reproducible on a robotic platform. The model proposed in this work, inspired in part by their observations (Xivry et al., 2013), aims at producing a model, suitable for a robotic implementation, able to combine feedback and predictive control.
From a robotic point of view, few works focus on the implementation of the smooth pursuit eye movement and none accounts for integrating the characteristics previously described. Shibata and colleagues suggested a control circuits for the integration of the most basic oculomotor behaviours (Shibata, Vijayakumar, et al., 2001) including the smooth pursuit eye movement. This model, based on the prediction of target dynamics did not consider a feedback control. The same consideration can be applied to the model of smooth pursuit and catch-up saccade (Falotico, Zambrano, et al., 2010) or occlusions (Falotico, Taiana, et al., 2009) implemented on the iCub robot. Also a model previously developed based on a neuro-controller (Vannucci et al., 2014), did not include a feedback control, but only prediction based on a Kalman filter.
The objective of this work is to investigate the applicability of smooth pursuit functional principles derived from neuroscience research on humanoid robots (Dario et al., 2005), in order to achieve a human-like visual pursuit able to: (i) use prediction for a zero-lag visual tracking, (ii) use a feedback based control for “unpredictable" target pursuit and (iii) combine these two approaches in order to guarantee a stable visual pursuit. In the next sections the model will be presented as well as its components in detail, together with two different implementations for the iCub humanoid robot. In order to validate the general model, two different inverse dynamics controllers (IDC) were implemented. In the past two approaches have been used to implement IDCs for oculomotor control: one that uses classic techniques to identify the eye plant parameters and build suitable controllers (Viollet and Franceschini, 2005) and one that uses neuro-controllers capable of adapting to the eye plant without having to identify its parameters (Lenz et al., 2008; Franchi et al., 2010). Thus, two different implementations are proposed: the first one, working in the velocity space,

uses a backstepping-based controller to generate velocity motor commands for the eye movements and the other one uses a bio-inspired neuro-controller to generate position motor commands for eye-neck coordinated movements.

2.2 Smooth Pursuit Model

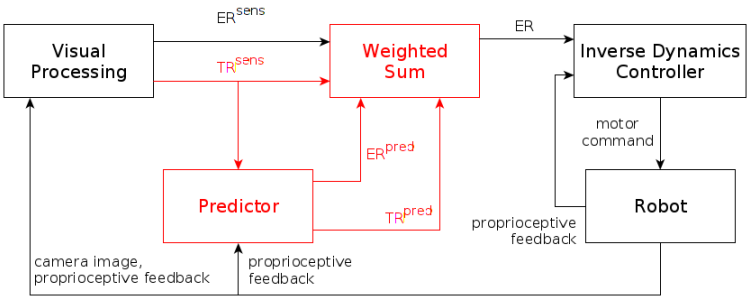


FIGURE 2.1: Model of the controller, with the predictive pathway and the switching mechanism highlighted in red.

The overall schema of the controller can be seen in Figure 2.1. The model consist of two main pathways: a sensory pathway (*Visual Processing*), where the camera image is processed to get sensory information about the target, and a predictive pathway (*Predictor*) where the sensory information is used to predict future states of the target.

Input coming from the sensory pathway (a target reference TR^{sens} and an error reference ER^{sens}) can be used to implement a classic feedback controller that is able to make the robot follow a moving target but with a certain delay.

On the other hand, input coming from the predictive pathway (a predicted target reference TR^{pred} and a predicted error reference ER^{pred}) can be used to anticipate the movement of the target and overcome the said delay, making the robot follow the target with zero lag. But the predictor will not perform well when the target velocity is unpredictable or when there is a change in the target motion. In such cases, the feedback controller will perform better. Thus, a proper switching mechanism is introduced to smoothly change between the input coming from the two pathways at the appropriate time (*Weighted Sum*).

In the end, the smoothed error reference ER is given to a controller capable of generating the appropriate motor commands for the Robot.

2.2.1 Predictor

In order to implement a model capable of predicting the target reference at future steps, a simple linear model has been chosen: Rosenblatt's single layer perceptron (Rosenblatt, 1958). To use this kind of model to predict future values of the signal a temporal sliding window of size d containing past values of the signal are given as an input along with the current value (Weigend, Huberman, and Rumelhart, 1990). A linear activation function was employed, as shown in Figure 2.2.

The output of the network at step t, for a given input signal x is:

$$out(t) = \sum_{i=0}^{d} x(t-d) \cdot w_d \tag{2.1}$$

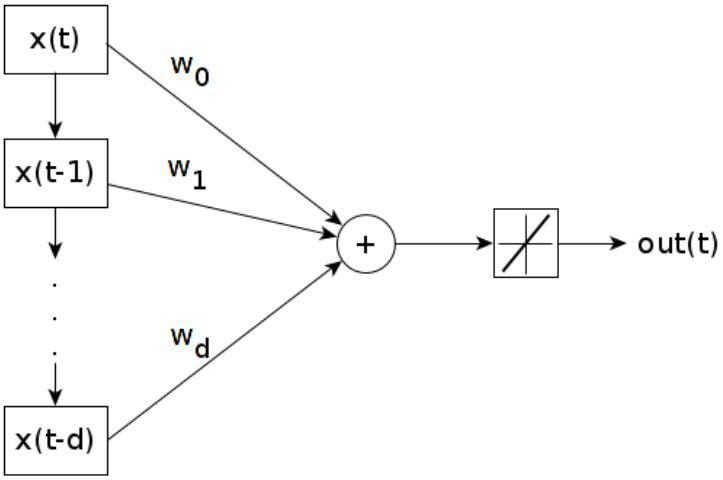


FIGURE 2.2: The single layer perceptron with the tap delay input.

The model was trained by an on-line version of the Widrow-Hoff rule (Widrow and Hoff, 1960), where the target value is the signal anticipated by p and η is the learning rate.

$$\Delta w(t) = \eta \cdot (x(t+p) - out(t)) \cdot x(t) \tag{2.2}$$

Using the on-line version of the learning algorithm gives the possibility of using the model without doing any previous training phase, as the perceptron will adapt the weights for the prediction during the execution phase. In order to move the signal towards 0 and thus facilitate learning, the mean observed value up to time t was subtracted from the signal.

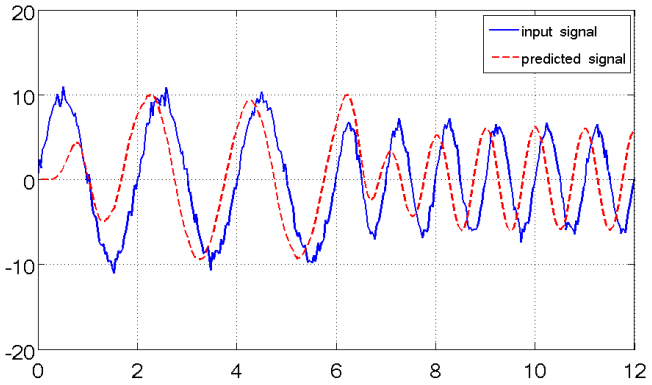


FIGURE 2.3: Prediction of a sinusoidal signal that changes amplitude and frequency, with added noise, simulated in MATLAB Simulink.

After choosing the parameters η and d, the model proved capable of predicting periodic signals, even with noise, as it is shown in Figure 2.3. The model was also able to quickly adapt to variations in frequency and amplitude of the signal and decay. In the overall control scheme the predictive model is used to predict future values of the target reference (TR^{pred}) starting from the current value of TR^{sens}. Then,

the predicted error reference is computed as follows, where is g is an appropriate function of the sensors:

$$ER^{pred}(t) = TR^{pred}(t) - g(sensors(t)) \tag{2.3}$$

While such a simple model is not capable to fully learn non-linear motion patterns and in fact can only interpolate the pattern with a linear regression, in oder to anticipate a target, predicting this linearized interpolation is enough, if the prediction is updated in an on-line fashion.

2.2.2 Weighted Sum

The control system must be able to automatically switch between the sensory and predictive pathways. In order to do so in a smooth manner, a weighted sum of the error references coming from the two pathways is performed:

$$ER(t) = (1 - \alpha(t))ER^{sens}(t) + \alpha(t)ER^{pred}(t) \tag{2.4}$$

where, for each t, $\alpha(t) \leq 1$. At each control step, α is chosen as a measure of how much the prediction has been accurate in the last 100 steps:

$$\alpha(t) = f(\max\{err(t), ..., err(t - 100)\}) \tag{2.5}$$

where, $err(t)$ is the error of the predictor at time t and f is a threshold function. In order to make the function α more smooth, f was implemented as a spline function, interpolating between these points:
$\{(0, 1), (0.04, 0.9), (0.08, 0.5), (0.184, 0.1), (0.4, 0)\}$ (Figure 2.4).

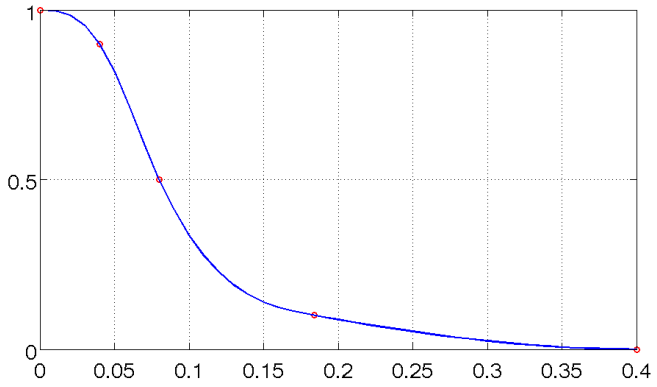


FIGURE 2.4: The α switching parameter as a function of the maximum prediction error.

Finally, to compute the error of the predictor at each step, its output is delayed by the same number of steps of the ahead prediction and then confronted with the input signal:

$$err(t) = \frac{|TR^{sens}(t) - TR^{pred}(t - p)|}{maxTR - meanTR} \tag{2.6}$$

where $maxTR$ and $meanTR$ are, respectively, the maximum observed value and the observed mean value of TR^{sens}.

As such, if the prediction is accurate enough the system will smoothly transition to take advantage of it, while, on the other hand, if the signal becomes unpredictable, it will revert to using only the sensory pathway. This can be seen in Figure 2.5, where it can be observed how the value of α changes according to the prediction accuracy, for both a predictable and a non-predictable input signals.

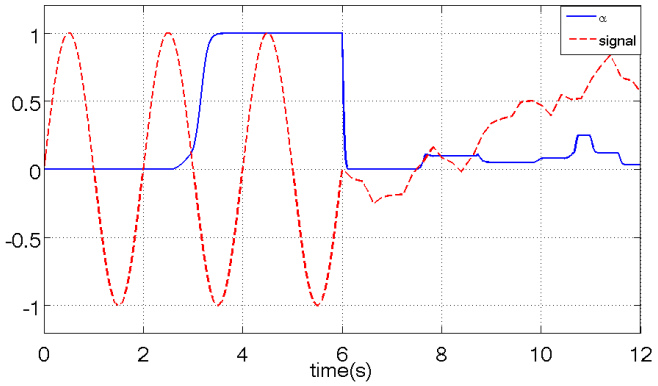


FIGURE 2.5: Prediction accuracy detection changes the value of α, by making it increase when the signal is predictable and decrease when it is not. Signals simulated in MATLAB Simulink.

2.3 Implementation

In order to validate the proposed model two different robotic implementations were tested on the iCub robot, using the simulator given with the iCub libraries. The iCub head (Beira et al., 2006) contains a total of 6 DOFs: 3 for the neck (pan, tilt and swing) and 3 for the eyes (an independent pan for each eye and a common tilt). The visual stereo system consists of 2 cameras with a maximal resolution of 640X480 pixels.

The first implementation has a backstepping-based controller as IDC receiving a velocity target reference to control the pan of the eyes through velocity motor commands, while the second has a neuro-controller receiving position target reference and to perform a full eye-head coordination through a joint position control.

A version of the backstepping-based controller was also implemented in MATLAB Simulink for a preliminary assessment of the smooth pursuit model. Moreover, the backstepping-based version was also employed on a physical iCub head to prove the effectiveness of the approach with real robotic platforms.

2.3.1 Backstepping-based controller as IDC

In this implementation the incoming reference is the velocity of the target on the horizontal axis and the error signal is the retinal slip, that is the speed of the target velocity projected in the eye reference frame. These values are computed by the *Visual Processing* module as follows:

$$TR^{sens}(t) = ER^{sens}(t) + \dot{\theta}(t) \tag{2.7}$$

where $\dot{\theta}$ is the current velocity of the pan joint. In the predictive pathway the retinal slip is computed from the predicted target velocity:

$$ER^{pred}(t) = TR^{pred}(t) - \dot{\theta}(t) \tag{2.8}$$

The smoothed error reference is then given to a controller, the objective of which is to cancel the dynamics of the eye plant. The system model of the eye plant is a second order transfer function given by

$$G(s) = \frac{1}{Js^2 + Bs} = \frac{k}{s^2 + as} \tag{2.9}$$

where J is the inertia of the system and B is the damp, with $k = 1/J$ and $a = B/J$. The system input is a velocity reference and its output is a position measurement. In particular, the velocity reference is the target velocity (TR) and its output is the position of the eye. For the system parameters, those estimated through an identification methodology on the simulated robot were used. The system can be expressed by the state-space function such as:

$$\begin{cases} \dot{\theta} = x_2 \\ \dot{x}_2 = ku - ax_2 \\ \theta = x_1 \end{cases} \tag{2.10}$$

The IDC that can negate the delay of such a plant is implemented using the *backstepping* technique, a design methodology for the construction of a robust feedback control law through a recursive construction of a Control Lyapunov Function (CLF). The proposed backstepping is based on (Kokotovie, 1992) and involves the inclusion of an integrator to the input of linear servo-motor model with a known feedback-stabilizing control law, and so the stabilizing approach is known as *integrator backstepping*. In particular, the *integrator backstepping* law provides suitable velocity commands and the possibility to exploit "good" non-linearities while "bad" non-linearities can be dominated by adding non-linear damping. The standard SISO model is:

$$\begin{cases} \dot{x} = f(x) + g(x)u = F(x,u) \\ f(0) = 0 \end{cases} \tag{2.11}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}$. The existence of a differentiable feedback control law $u = \Gamma(x)$, with $\Gamma_i(0) = 0$ and of a CLF $\mathbf{V}(x) : \mathbb{R}^n \to \mathbb{R}$ such that $L_F\mathbf{V}(x) \leq 0$ are assumed. Hence, the augmented system with an integrator is given by:

$$\begin{cases} \dot{x} = f(x) + g(x)\nu = F(x,\nu) \\ \dot{\nu} = u \end{cases} \tag{2.12}$$

where $f(x)$ is the drift function of the model and u is the input command. If $L_F\mathbf{V}(x)$ is negative definite, then the CLF becomes:

$$\mathbf{V}_e(x,\nu) = \mathbf{V}(x) + \frac{1}{2}[\nu - \Gamma(x)]^2 \tag{2.13}$$

where the function $\mathbf{V}(x)$ is the possible CLF that fulfils other constraints. A feedback control $u = \Gamma(x,\nu)$ exists such that the equilibrium point $(x,\nu) = (0,0)$ is globally asymptotically stable. The possible control that satisfies the assumption (2.13) is:

$$u = -\lambda(\nu - \Gamma(x)) + \frac{\partial\Gamma(x)}{\partial x}g(x)\nu - \frac{\partial\mathbf{V}(x)}{\partial x}g(x) \tag{2.14}$$

with $\lambda > 0$. This control can be replaced in the CLF directional derivative obtaining:

$$L_F \mathbf{V}_e(x) = L_F \mathbf{V}(x) - \lambda[\nu - \Gamma(x)]^2 \tag{2.15}$$

where $L_F \mathbf{V}(x)$ is the directional derivative of the possible other constraints. Then, the overall CLF is defined negative. Therefore, the expanded Lyapunov candidate is:

$$\mathbf{V} = \frac{1}{2}(ER)^2 \tag{2.16}$$

where in this case the retinal slip is computed as $ER = (TR - x_2)$. The time derivative of Lyapunov candidate (2.16) can be derived as:

$$\dot{\mathbf{V}} = (ER)(\dot{ER}) = ER(\dot{TR} - ku + ax_2) \tag{2.17}$$

and after some easy algebraic computations, the control input for the motor can be rewritten as listed below:

$$u = \frac{1}{k}[\dot{TR} + ax_2 + \lambda(ER)] \tag{2.18}$$

The values of λ is strictly positive and is defined on the basis of the desired convergence velocity of the algorithm. In this case, the control law does not include the derivative terms as in (2.14) as it could represent a possible noise source. The absence of derivative terms is balanced by the presence of the constant λ. Furthermore, replacing (2.18) in the CLF derivative (2.17) the asymptotic stability of Lyapunov candidate is demonstrated:

$$\dot{\mathbf{V}} = -\lambda(ER)^2 \leq 0. \tag{2.19}$$

2.3.2 Neuro-controller as IDC

In this implementation, the pieces of information gathered from the *Visual Processing* are the target position in the 3D space as the target reference and the difference between the target position and the gaze fixation point. The current gaze fixation point (gfp) was computed by using a direct kinematics function K, so, if θ are the values of the head joints, then, for the sensory and predictive pathways:

$$ER^{sens}(t) = TR^{sens}(t) - gfp(t) \tag{2.20}$$
$$ER^{pred}(t) = TR^{pred}(t) - gfp(t) \tag{2.21}$$
$$gfp(t) = K(\theta(t)) \tag{2.22}$$

The smoothed ER signal is given to a neuro-controller capable of moving the current gaze fixation point to compensate for the error in the 3D space with a coordinated movement of both the robot eyes and neck joints, including eye vergence. This controller, presented in (Vannucci et al., 2014) as an improved version of (Asuni et al., 2005), makes use of two different bio-inspired neural models: a Growing Neural Gas for associative learning of motion patterns and a biologically recurrent neural network that generates the position motor commands. The neuro-controller employed is depicted in Figure 2.6.

The capability of this controller is to move the current gaze fixation point (gfp) of the robot towards the target point. The core part of this controller is the Sensory-Motor Map, which is a network able to respond to a stimulus by activating some specific units. This is similar to the functioning of the somatosensory cortex in the

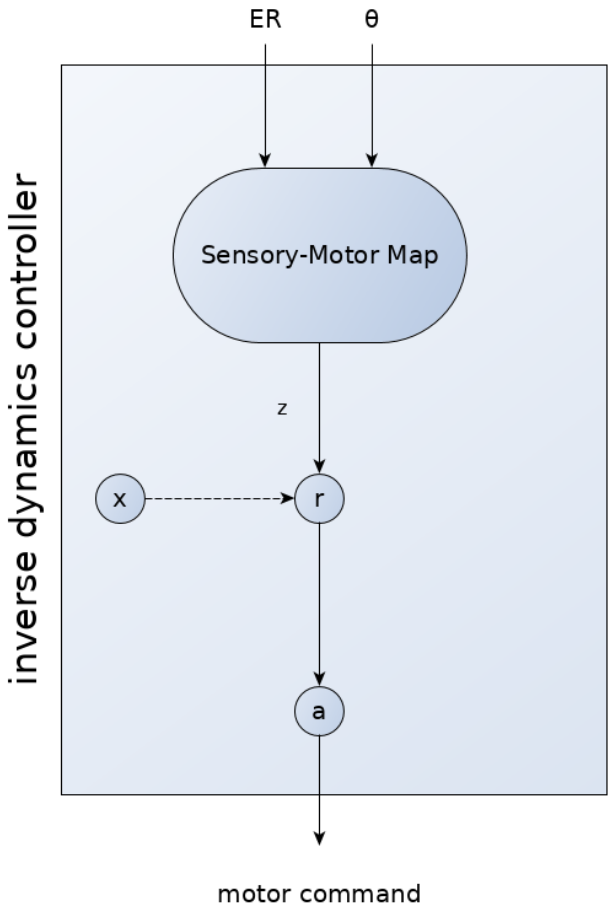


FIGURE 2.6: The neuro-controller employed as an inverse dynamics controller.

human brain: a stimulus reaches the cortex and the section dedicated to respond to that stimulus activates. In this case, the stimulus, or input, is the normalized error reference combined with the proprioceptive feedback:

$$p(t) = \left(\frac{ER(t)}{||ER(t)||_2}, \theta(t) \right) \tag{2.23}$$

This map is represented by a neural model capable of learning how to distinguish between different inputs, the Growing Neural Gas (Fritzke, 1995) (GNG). The Neural Gas model (Martinetz, Schulten, et al., 1991) (NG) is similar to a Kohonen's map (Kohonen, 2001), but has the benefit of not having a fixed topology. The main difference between the GNG and the NG is that in the first model the network of units grows dynamically during the training phase, instead of being of a fixed size decided beforehand. The output of such network is the activated unit, or winning unit:

$$gng(p) = \arg\min_{s \in A} ||p - w_s||^2 \tag{2.24}$$

where A is the set of units and w_s is the weight vector of the s-th unit. For each unit s, its topological neighbourhood is defined as

$$N_s = \{u \in A | (s,u) \in E\} \tag{2.25}$$

where E is a dynamically built subset of $A \times A$.

After a stimulus is given to the Sensory-Motor Map, this will then respond by generating the motor commands that will move the gaze fixation point towards the desired position. This is done by passing the activation signal through a series of units specifically trained by a motor babbling algorithm to generate these motor commands. The r units represent the agonist and antagonist activations for each motor. Thus, the number of these units is double the number of the actual actuators so that the couple r_{2i}, r_{2i+1} are the agonist and antagonist units for the i-th motor, also called r_i^E, r_i^I, which stand for the excitatory and inhibitory stimuli for the i-th actuator. Their activation is computed as:

$$r_i = x_i + z_{wi} + \frac{\sum_{k \in N_w} \nu z_{ki}}{|N_w|} \tag{2.26}$$

where w is the index of the winning unit, z_{ki} is a weight coming from the k-th unit of the GNG, ν is a proper constant and x_i is a random activation coming from x units population, only present during the training phase. The a units are responsible for the generation of the actual motor commands that have to be sent to the actuators, starting from the excitatory and inhibitory stimuli coming from the r population. At each control step the new output values are computed in terms of a difference between current and previous step values:

$$\frac{\mathrm{d}a_i}{\mathrm{d}t} = \epsilon(||o(t+j) - gfp(t)||_2) \cdot (r_i^E - r_i^I) \cdot g(r_i^E, r_i^I, a_i(t-1)) \tag{2.27}$$

where $\epsilon(d)$ is a function of the distance between the target and the current gaze fixation point defined as

$$\epsilon(d) = \begin{cases} v \cdot d & \text{during the execution phase} \\ \epsilon & \text{during the training phase} \end{cases} \tag{2.28}$$

with an appropriate speed parameters v and ϵ, and

$$g(r^E, r^I, \psi) = \begin{cases} \psi_{max} - \psi & \text{if } (r^E - r^I) \geq 0 \\ \psi & \text{if } (r^E - r^I) < 0 \end{cases} \tag{2.29}$$

where ψ_{max} is the maximum angle of the joint.

The training for this controller is performed in two phases, both unsupervised:

- in the first phase only the sensory-motor map is trained;
- in the second, the motor babbling learning is performed, the newly trained map is used in the global architecture to train the z connections.

The z connections are trained by using Grossberg's Outstar Law (Grossberg, 1976):

$$\frac{\mathrm{d}z_{ki}}{\mathrm{d}t} = \gamma \cdot c_k \cdot (-\delta z_{ki} + r_i) \tag{2.30}$$

where γ is the learning rate and δ is a decay parameter. At a given step, only the connections going out from the winner units and its neighbours are updated, otherwise the update would not be related to the recently performed movement. This is done by defining:

$$c_k = \begin{cases} 1 & \text{if } k \text{ is the winner unit } w \text{ or } k \in N_w \\ 0 & \text{else} \end{cases} \tag{2.31}$$

For the training of the neuro-controller, the process started by training the sensory motor-map, namely the growing neural gas. To do this a training set of 100000 input patterns was created and then given in input ten times to the GNG. After the training ended, the resulting network had 3695 units.

The selection of the hyperparameters for the controller $(\gamma, \delta, \epsilon, \nu)$ is critical and greatly impacts the performance. To appropriately choose such parameter, a model selection procedure was implemented. During this procedure, the performance of the resulting model was measured by computing its mean speed of reaching on a training set D of 20 target points:

$$v = \frac{1}{|D|} \sum_{i \in D} \frac{s_i}{t_i} \tag{2.32}$$

where s_i the starting distance of the target and t_i the number of control step used to reach it. The model selection was performed on these values for the hyperparameters:

- $\gamma \in \{0.005, 0.01, 0.015, 0.02\}$;
- $\delta \in \{0.0005, 0.001, 0.0015, 0.002\}$;
- $\epsilon \in \{0.1, 0.3, 0.05\}$;
- $\nu \in \{0.1, 0.3, 0.5\}$.

The best results on the validation set were obtained by a model with a mean speed of 0.055 meters/step and the following values for the hyperparameters:

$$\gamma = 0.005 \quad \delta = 0.002 \quad \epsilon = 0.1 \quad \nu = 0.1$$

All the training phase was done using the iCub simulator.

2.4 Results

To assess the effectiveness of the adaptive smooth pursuit model, an initial set of preliminary assessment was performed by implementing the backstepping-based controller on MATLAB Simulink. This allowed testing in a more controlled environment, where sensor noise could be adjusted to various intensities or nullified.

In order to test the robotic implementations, two kinds of tests have been performed on both of them.

In the first test, the reference signal switches from a sinusoidal wave to another, after a certain period of time (Figure 2.7). This test is performed in order to show the capability of the model to adapt to a change in frequency and amplitude of the signal.

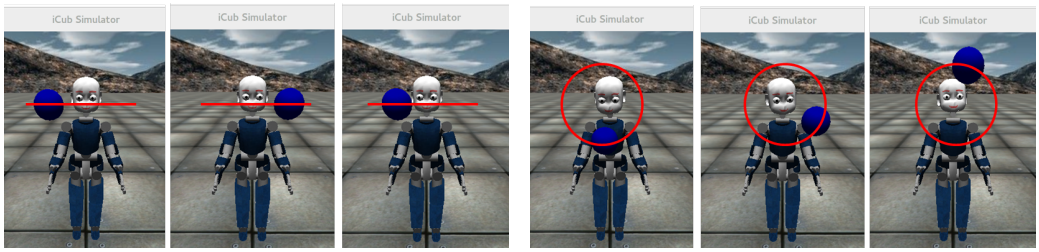


FIGURE 2.7: Trial executions showing the iCub robot pursuing a target. Left images show the horizontal task for the first implementation, while the right ones show the circular motion used in the second implementation. The red line shows the target motion.

In the second test, the reference signal switches to a random, unpredictable one. This test is performed in order to demonstrate that the model is still able to follow an unpredictable signal even if with some delay.

For both implementations the tests were performed on the iCub simulator where a ball was moved inside the simulated environment and then tracked from the camera images. It should be noticed that the objective of this work is not to provide a comparison among the two controllers performances, but to asses the model through these controller implementations.

Finally, in order to ensure the usefulness of the approach for realistic scenarios, tests of the backstepping-based version of the controller were performed on a physical iCub robotic head. In this case a pendulum whose length could be changed was employed to give a periodic motion to the target.

In all cases, only information from one camera was employed, thus binocular vision was not exploited. Nonetheless, while in the backstepping based implementation the eyes perform the same motion and only one is actually centring the target, the neuro-controller is capable of controlling the eyes vergence and thus to centre the target with both eyes.

2.4.1 Preliminary validation

In this experiments, performed on the Simulink implementation, the backstepping-based controller had to follow a sinusoidal motion target with a peak-to peak amplitude of 16 degrees for 7 seconds. The performance of the model were tested at various angular frequencies ($1/8\pi rad/s$, $1/4\pi rad/s$, $1/2\pi rad/s$, $\pi rad/s$, $3/2\pi rad/s$, $2\pi rad/s$) and noise levels (0%, 4%, 8%). It should be noticed that keeping the motion peak-to-peak amplitude fixed (16deg) and incresing the frequency result in an amplitude changing of the peak-to-peak target velocity from 6.3 deg/s (for the angular frequency of $1/8\pi rad/s$) to 100 deg/s (for the angular frequency of $2\pi rad/s$). Figure 2.8 and Figure 2.9 show the performance of the controller with two different noise levels (0%, 8%). The eye velocity is completely aligned in time after two seconds (when the system starts “trusting" the predictor output) for the trial with no noise and the error is almost nullified ($|RS| < 0.15$deg/s). In the other trial, the controller needs more time to reduce the error and, in this case, the retinal slip remains higher than 2 deg/s. To evaluate the performance trend, the Mean Squared Error (MSE) during the last second of the trial for the considered conditions of angular frequency and noise levels was computed and compared at different frequencies (Figure 2.10).

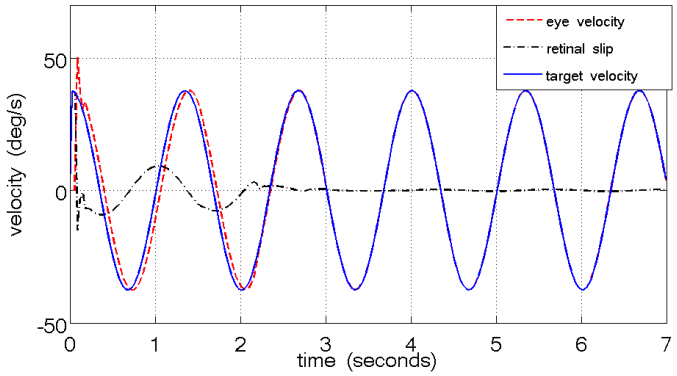


FIGURE 2.8: Performance of the model during a task with a sinusoidal motion with an angular frequency of $3/2\pi$ rad/s.

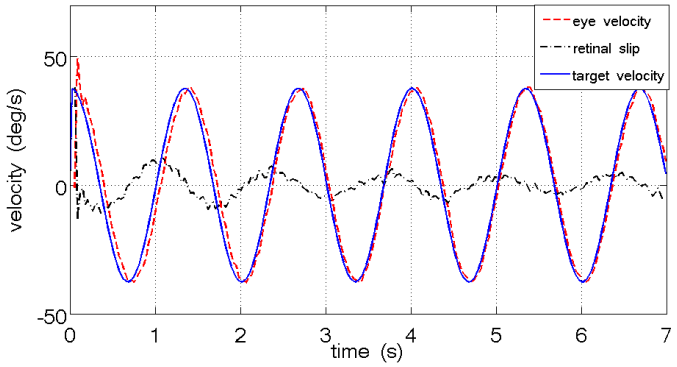


FIGURE 2.9: Performance of the model during a task with a changing target motion with an angular frequency of $3/2\pi$ rad/s and 8% noise level. The target dynamics is showed without the added noise

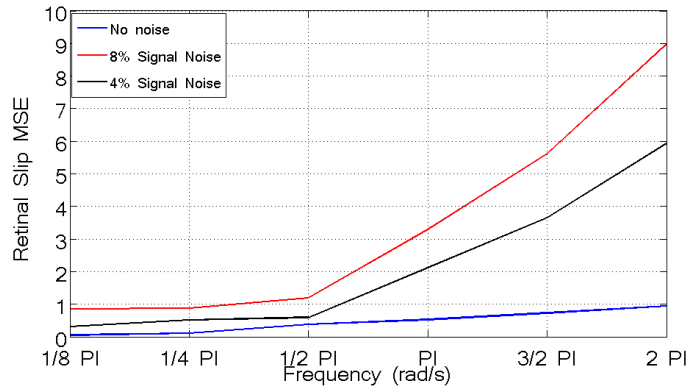


FIGURE 2.10: MSE for the considered trials. In case of trial with no noise the MSE is less than 1.

2.4.2 Results for the Backstepping-based controller as IDC

Using this implementation, the two kinds of tests were performed by moving the ball on the horizontal axis, with a sinusoidal motion. The retinal slip and the joint velocity are computed by applying a least-squares algorithm (Janabi-Sharifi, Hayward, and C.-S. Chen, 2000). The parameters used for the predictor in these tests were: prediction step $p = 5$, tap delay $d = 50$ and learning rate $\eta = 0.5$. The parameters for the controller were found empirically as: $k = 25.14$, $a = 30.43$, $\lambda = 60$.

Results for the signal switching task can be seen in Figure 2.11. The initial frequency of the movement was 0.25Hz with an amplitude of 0.1m, and it can be observed that the value of α increases, meaning that the prediction becomes more accurate. The maximum peak-to-peak value of the retinal slip decreases from 8 deg/s to 3 deg/s. When the motion changes in both frequency and amplitude (0.125Hz and 0.15m), after 26 seconds, the value of α suddenly decreases towards zero, resulting in an higher retinal slip. After a certain period of time, the prediction is again accurate, therefore the value of α increase again and the eye velocity aligns again with the target one, resulting in an maximum retinal slip peak-to-peak value of 3 deg/s.

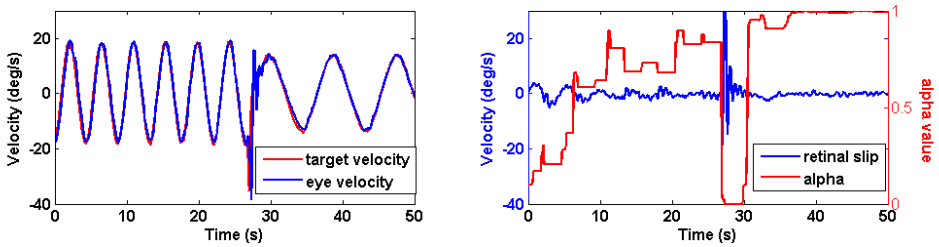


FIGURE 2.11: Results for a frequency and amplitude changing task, in the backstepping-based implementation. The left plot shows the reference velocity and the eye velocity, while the right plot shows the velocity error and the value of α. Values of α close to 1 correspond to a strong contribution of the prediction in the control loop, values close to 0 correspond to a pure feedback control.

During the initial phase of the second task, the motion has the same frequency and amplitude as the previous one, but after 26 seconds it changes to a random one (Figure 2.12). After the change, the value of α suddenly decrease towards zero, but the eye is still able to reach the target velocity, even with some delay. During this second phase, the mean peak-to-peak value of the error is 6.86 deg/s.

2.4.3 Results for the Neuro-controller as IDC

For the neural implementation the test were performed by moving the ball on two axes, both the horizontal and vertical, with a sinusoidal motion on each. The parameters used for the predictor in these tests were: prediction step $p = 5$, tap delay $d = 50$ and learning rate $\eta = 0.5$.

The same values for the frequency and amplitude of the movements used for the other implementation were used for this one.

Figure 2.13 shows the results for the first task. During the first phase of the motion the position error decreases from a maximum peak-to-peak amplitude of 0.04m to 0.2m on both axes. After the change of frequency and amplitude at 26s, the

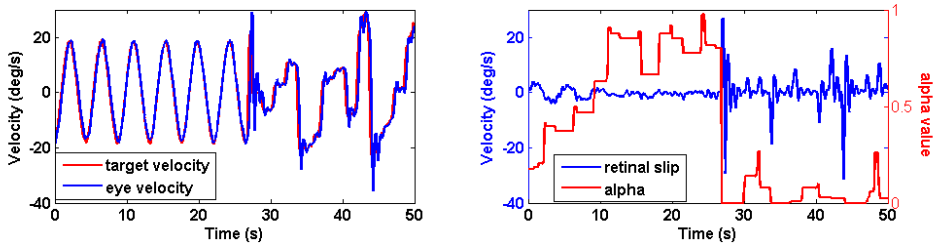


FIGURE 2.12: Results for a task with a switch to a random signal, in the backstepping-based implementation. The left plot shows the reference velocity and the eye velocity, while the right plot shows the velocity error and the value of α.

value of α decreases and then grows again, meaning that the predictor adjusted for the new signal. The error then decreases again under 0.02m on both axes.

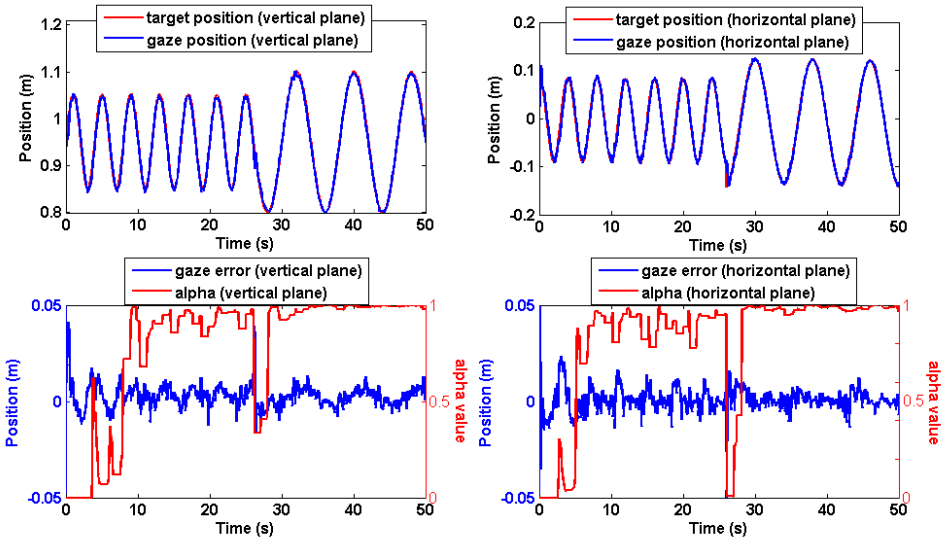


FIGURE 2.13: Results for the neural implementation on a signal switching task. The top plots show the reference position and the gaze position on both the horizontal and the vertical axes, while the bottom plots show the position error and the value of α for both axes.

Also in this implementation the second task starts with an initial motion phases equal to the previous one (Figure 2.14). Again, after the change, the value of α suddenly decreases, but the robot is still able to follow the target, with a maximum peak-to-peak amplitude of the error of 0.05m on each axis.

2.4.4 Results on the physical robot

Due to the higher sensory and actuation delays of the physical robot, a higher prediction step than the one used in simulation had to be employed ($p = 12$), with a tap delay of $d = 40$ and a learning rate of $\eta = 0.25$. Again, the parameters for the controller were found empirically as: $k = 57.41$, $a = 40.34$, $\lambda = 80$.

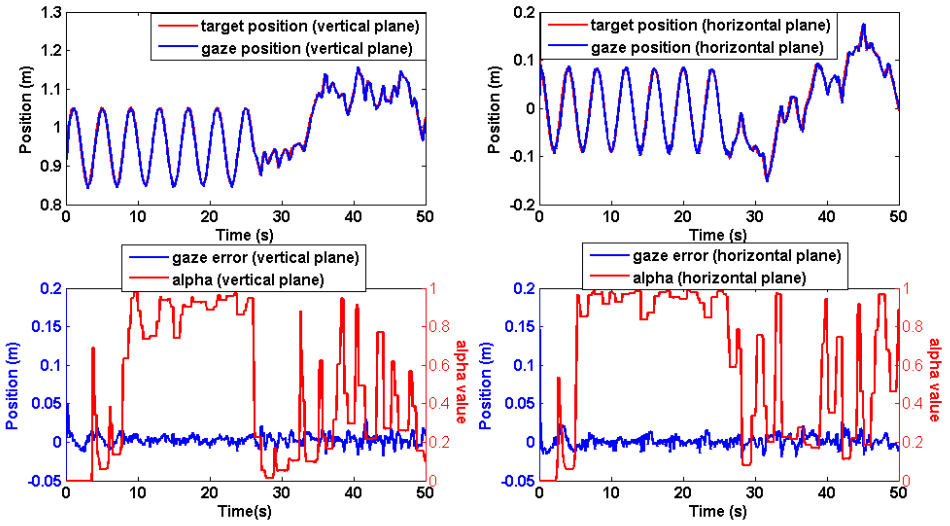


FIGURE 2.14: Results for the neural implementation task with a switch to random motion. The top plots show the reference position and the gaze position on both the horizontal and the vertical axes, while the bottom plots show the position error and the value of α for both axes.

An example of a basic trial with a frequency of swings of circa 0.4Hz can be observed in Figure 2.15. Due to the lower learning rate and higher prediction step employed, the predictor takes more time to learn the target motion. This, in turn, affects the switching mechanism that is slow in trusting the prediction and switch to the predictive pathway. Nonetheless, the model is able to employ the predictive information to cancel the delay and reduce the retinal slip below 4deg/s. It should be noticed that the amplitude of the motion during these trials reduces over time due to air resistance of the pendulum.

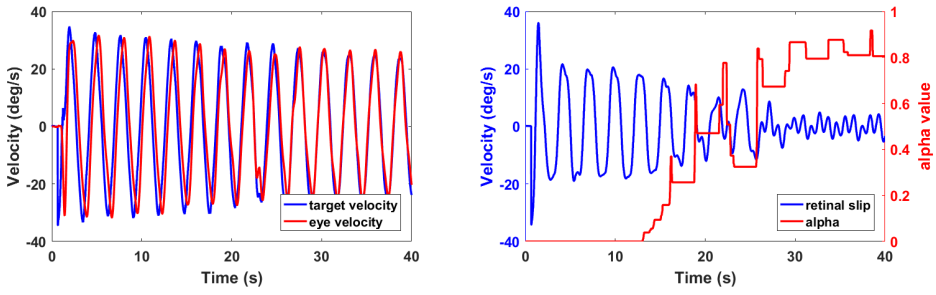


FIGURE 2.15: Results for a constant frequency task, in the backstepping-based implementation, on the real robotic platform. The left plot shows the reference velocity and the eye velocity, while the right plot shows the velocity error and the value of α.

Much like in simulation, the adaptive switching mechanism works on the physical robot, as show by the experimental trial reported in Figure 2.16. In this test the motion started with a frequency of 0.5Hz and then it was manually switched to 0.3Hz, while also changing the amplitude. It can be observed that the model still take more time than in simulation to switch to the predictive pathway, but it can

cancel the delay, immediately recognize the change in dynamics of the target and adapt relatively fast to the new motion.

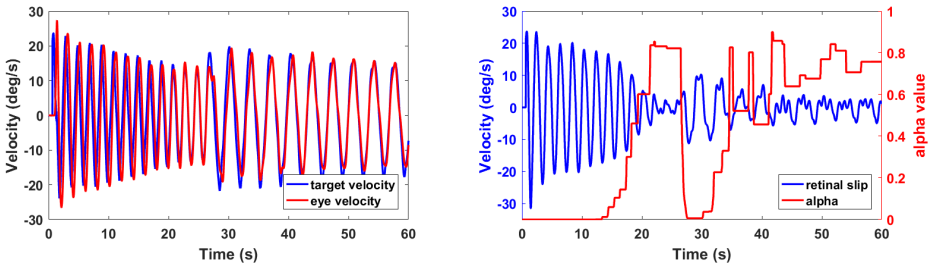


FIGURE 2.16: Results for frequency and amplitude changing task, in the backstepping-based implementation, on the real robotic platform. The left plot shows the reference velocity and the eye velocity, while the right plot shows the velocity error and the value of α.

2.5 Conclusions

In this chapter, a model able to integrate the major characteristics of visually guided and predictive control of the smooth pursuit was presented. This model is inspired by neuroscientific findings and make use of neural methods to predict the dynamics of the target motion. The model also include an automatic switching mechanism that let the system automatically adapt to changes in the target motion. To validate the general model, two different controllers were implemented. The first one, working in the velocity space, uses a backstepping-based controller to generate velocity motor commands for the eye movements. The other one is based on a bio-inspired neuro-controller able to generate position motor commands for eye-neck coordinated movements. The obtained results, tested on on the iCub simulator and on a physical roboti, show that both the proposed implementations can guarantee a stable visual pursuit switching from predictive to feedback control.

Chapter 3

Prediction as sensory-motor anticipation: gaze stabilization

Contrary to what was presented in Chapter 2, where a model based on the prediction of external dynamics was shown, in this chapter prediction of sensory feedback and internal states will be investigated. This will be achieved by the development of a gaze stabilization model based on cerebellar internal models capable of providing sensory-motor anticipation, that is prediction of proprioceptive feedback and/or internal states.

3.1 Introduction

Gaze stabilization is fundamental in everyday activities. The way it compensates the movements of the other part of the body as the torso, especially during basic movements like walking or running, is crucial to give a stable reference frame for the two essential perceptual systems for detection of self-motion relative to space: the visual and vestibular systems. Vision is the most useful sensor for many animals to provide information about the surrounding environment. The vestibular information operates to create an inertial guidance system determining the spatial orientation in order to coordinate movements and balance. Two kind of reflexes, which rely on the output of the inertial system and are used to keep the image stable on the retina, exist: the vestibulo-ocular reflex (VOR), which stabilizes the visual axis to minimize retinal image motion and the vestibulocollic reflex (VCR), which stabilizes the head in space through the activation of the neck musculature in response to vestibular inputs.

3.1.1 VOR - From neuroscientific evidences and models to robotic controllers

The VOR compensates for head movements that would perturb vision by turning the eye in the orbit in the opposite direction of the head movements (G. Barnes, 1993). VOR works in conjunction with the opto-kinetic reflex (OKR), which is a

This chapter has been adapted from the articles:

Vannucci, L., E. Falotico, S. Tolu, V. Cacucciolo, P. Dario, H. H. Lund, and C. Laschi (2017). "A comprehensive gaze stabilization controller based on cerebellar internal models". In: *Bioinspiration & biomimetics* 12.6, p. 065001.

Vannucci, L., E. Falotico, S. Tolu, P. Dario, H. Lund, and C. Laschi (2016). "Eye-head stabilization mechanism for a humanoid robot tested on human inertial data". In: *Lecture Notes in Computer Science* 9793, pp. 341–352.

Vannucci, L., S. Tolu, E. Falotico, P. Dario, H. Lund, and C. Laschi (2016). "Adaptive gaze stabilization through cerebellar internal models in a humanoid robot". In: *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*. Vol. 2016-July, pp. 25–30.

feedback mechanism for moving the eye at the same speed as the observed scene. Together they keep the image stationary on the retina, with VOR compensating for fast movements and OKR for slower ones (Schweigart et al., 1997). The VOR involves six extraocular muscles, each pair acts around a single rotation axis. When a rotation of the head is detected, the semicircular canals get activated by head rotation and send their impulses via the vestibular nerve, through the vestibular nuclei, to the extraocular muscles (an inhibitory signal to the muscles on one side and an excitatory signal to the muscles on the other side). The result is a compensatory movement of the eyes. The neural control circuitry of the horizontal VOR has been studied intensively since it involves only three neuron reflex arc and the cerebellar flocculus (Ito, 1984) which is also known to be involved in the OKR. The VOR is an adaptive mechanism which changes with experience. The vestibulocerebellum is considered to be responsible for this learning. Several studies confirm that the cerebellum is essential for motor learning to correct the VOR in order to ensure accurate eye movement. To test this hypothesis, Ito and colleagues (Ito, Nisimaru, and M. Yamamoto, 1977) studied the influence of a stimulation applied to the inferior olivary or to the optic tract to the vestibulo-ocular reflex. In healthy rabbit, an olivary stimulation induced a depression of the discharge sent by the semicircular canals. In parallel with the electrophysiological studies, authors tested the effect of a lesion on the VOR. Using spectacles that reversed left and right direction on cats, Robinson (D. Robinson, 1976) showed that cats are able to adapt their VOR gain to achieve a stable vision. When the flocculus, paraflocculus and some parts of the vermis were removed, Robinson (D. Robinson, 1976) reported no more long- or short-term adaptation. An interesting hypothesis on cerebellar motor learning, based on an experimental study of the rabbit cerebellum, was proposed by Ito (Ito, 2000) who followed the Marr-Albus model (Marr and Thach, 1991; Albus, 1971). According to Ito, the adjustment of the relative strengths of the direct excitatory and indirect inhibitory pathways adaptively modulates the gain of the VOR. Ito asserted that the modulation could be influenced by messages of retinal image slip and the role of the cerebellum is to store the motor memory for the changes in VOR gains. Miles and Lisberger (Miles and Lisberger, 1981) proposed an alternative model. They stated that the role of the cerebellum was not to store memories but rather to compute an instructive signal to guide the plasticity process.
One of the first neuroscientific models of VOR has been proposed by Schmid et al. (Schmid, Stefanelli, and Mira, 1971) based on human recordings with the aim to compare the frequency response of their model to the human behavior. Galiana and Outerbridge (Galiana and Outerbridge, 1984) published a bilateral model of the VOR based on the anatomy of the main neuronal connections present in the central VOR pathway. The model can reproduce the activity of the different populations of neurons when a subject is rotating in the dark. Gomi and Kawato (Gomi and Kawato, 1992) proposed a model based on the simultaneous adaptation of VOR and OKR, and compared it with the biological data. Finally, Green and Angelaki (Green and Angelaki, 2004) proposed model based on a neural network that combines the sensory information from otoliths and semicircular canals simulating the response along the three rotational axes (yaw, pitch and roll). Other models focus on the visual-vestibular interaction ((Lau et al., 1978) (Schmid, Buizza, and Zambarbieri, 1980) (G. Barnes, 1993). Lau and colleagues (Lau et al., 1978) published a VOR model with a linear interaction between the vestibular and the visual component. With this simple model, they simulated the behaviour of the VOR in the dark, in the light, with a head-fixed target and with a target oscillation added to the scillating chair. The model of Lau et al. (Lau et al., 1978) was built to reproduce all

the experimental conditions for a single oscillation frequency of 0.05Hz. After that, Schmid et al. (Schmid, Buizza, and Zambarbieri, 1980) updated the model of Lau et al. (Lau et al., 1978) and integrated non-linear elements to better represent the behaviour of the visual-vestibular interaction on a larger range of target velocities. Barnes (G. Barnes, 1993) proposed a model that includes three loops to reproduce the visual modulation during pursuit and the non-visual modulation of the VOR. In this model, a central control triggers either an enhancement or a decrease of the VOR response to model a volitional control of the VOR. All these models aim to validate human experiments or try to replicate the neural circuits involved in this tasks, but are not suitable for the implementation on robotic platforms.
Two biological models of the VOR have been proposed and tested on robotic platforms. Shibata and Shaal (Shibata, Vijayakumar, et al., 2001) propose to use a feedback error learning strategy in conjunction with a receptive field weighted regression algorithm to implement the VOR and its internal model on a humanoid robot. Porrill and colleagues (Porrill, Dean, and Stone, 2004) presented a model for the VOR that is capable of working on three rotational axes and that includes an internal model and provide an implementation on Matlab. This model has also been implemented on a robotic platform simulating the eye movements (Lenz et al., 2008) . In robotics literature some other controllers inspired by the VOR can be found (Viollet and Franceschini, 2005; Franchi et al., 2010). Viollet and Franceschini developed a gaze controller that includes a feedforward component inspired by the VOR, but lacking any sensory-motor corrective module. Finally, in (Franchi et al., 2010), a decorrelation model replicating the VOR, that is implemented with a Recursive Least Square algorithm, is shown to be working on a robotic simulator. Panerai and colleagues presented a bio-inpired controller based on an inertial sensory apparatus and images of space-variant resolution (Panerai and Sandini, 1998; Panerai, Metta, and Sandini, 2000; Sandini, Panerai, and Miles, 2001). Among all these controllers, only Shibata and Schaal (Shibata, Vijayakumar, et al., 2001) replicated the OKR mechanism and showed the cooperation between ocular movements.

3.1.2 VCR - From neuroscientific evidences and models to robotic controllers

The VCR stabilizes the head based on the inertial input space by generating a command that moves the head in the opposite direction to that of the current head-in-space displacement. When the head is rotated in the plane of a semicircular canal, the canal is stimulated and the muscles are activated. Thanks to this stimulation, a compensatory rotation of the head along the same axis is produced. Each canal activation produces an appropriate reflex response. The VCR controls a complex musculature that includes more than 30 muscles controlling pitch, roll and yaw rotations (B. W. Peterson et al., 2001). The system could be considered underdetermined because there are more muscles than rotation axes. One consequence of this property is that the same head movement can be produced by the activation of different muscle patterns: this is the case for voluntary head movements, but not for the VCR where a particular head motion is related to a stereotyped muscle activation pattern (B. W. Peterson et al., 2001).
The neuroscientific studies conducted on human patients during locomotion evidence the contribution of the VCR for the head stabilization. During a normal straight walking the head moves mainly in the sagittal plane, rotating around the pitch axes in a range of $0.3 - 8^{\circ}$ and translating between $0.8 - 9$ cm (Pozzo, Berthoz, and Lefort, 1990; J. Bloomberg et al., 1992). During fixed-gaze treadmill walking,

that is at fixed velocity, the coordination between the head and the trunk is dependent on the events in a gait cycle (Mulavara, Verstraete, and J. J. Bloomberg, 2002), although the trunk already damps many oscillation acting as a low pass filter (Kavanagh, Barrett, and Morrison, 2006), the role of the reflexes is still essential. The neural pathways mediating this reflex are as yet uncertain. Despite this, the involvement of the cerebellum in the vestibulo spinal reflexes, including the VCR, is well known (Cullen and Roy, 2004; Manzoni et al., 1994). For a complete review about the role of vestibular system in posture control and involvement of the cerebellum in stabilization tasks refer to (Cullen, 2012). Task involving visual fixation lead to an improvement of head-in-space stabilization in humans (Goldberg and Cullen, 2011) due to increased VCR contributions (Forbes, Bruijn, et al., 2013). Rather it is still unclear whether a suppression of the VCR occurs in voluntary head movements (Forbes, Siegmund, et al., 2014). Although there are many physiological models of the VOR and many robotic implementations, it is not really the same for the VCR. The existent models have incomplete characteristics or are not suitable for a robotic implementation. The current state of art provides two main important models for the aforementioned reflex. Both consider the VCR as a simple *negative feedback*, but recent studies (Forbes, Bruijn, et al., 2013) demonstrates the engagement of the central nervous system in the modulations of feedback gains. This means that the simple feedback is not enough to explain the mechanisms involved in head stabilization and that some feedforward contribution is present.
A model of VCR has been proposed by Peng (Peng, Hain, and B. Peterson, 1996) in 1996. They presented a control model of yaw head rotations during perturbations of the torso, which for the first time combine a model of the human head with neural feedback controllers representing the vestibulocollic and the cervicocollic reflexes. The parameters of the model are extracted directly from anthropomorphic, biomechanical and physiological studies. The model proposed by Goldberg and colleagues (Goldberg and Cullen, 2011) is nearly the same as the previous one but in addition there is the contribution of the voluntary control.
In robotics, some head stabilization models already exist. Yamada and colleagues (Yamada, Mori, and Hirose, 2007) proposed a method for controlling the neck of a snake-like robot in order to stabilize the head against undulations. The controller is based on the rejection of the disturbance of the body on the head using a continuous model. The model proposed by Marcinkiewicz (Marcinkiewicz et al., 2009), implemented on the AIBO robot, uses a machine learning algorithm able to learn how to compensate for head rotations in the absence of stabilization mechanisms. Gay and colleagues (Gay, Santos-Victor, and A. Ijspeert, 2013) proposed a head stabilization system for a bipedal robot during locomotion controlled by the optical flow information. It is based on Adaptive Frequency Oscillators to learn the frequency and phase shift of the optical flow. Although the system can successfully stabilize the head of the robot during its locomotion, it does not take in consideration the vestibular inputs.

The closest to the neuroscientific findings of the VCR are the works proposed by Kryczka and colleagues (Kryczka, Falotico, Hashimoto, H. Lim, et al., 2012; Kryczka, Falotico, Hashimoto, H.-O. Lim, et al., 2012; Falotico, Cauli, Hashimoto, et al., 2012). They proposed an inverse jacobian controller (Kryczka, Falotico, Hashimoto, H. Lim, et al., 2012; Kryczka, Falotico, Hashimoto, H.-O. Lim, et al., 2012) based on neuroscientific results (Falotico, Laschi, et al., 2011) and an adaptive model based on a feedback error learning (FEL) (Falotico, Cauli, Hashimoto, et al., 2012) able to compensate the disturbance represented by the trunk rotations. A comparison between these models of head stabilization is provided in (Falotico, Cauli, Kryczka, et al.,

2017).

3.1.3 Objectives and rationale

All the presented models try to reproduce specific aspects of the gaze stabilization behaviour, but none of them can provide a comprehensive model of gaze stabilization, integrating eye stabilization (OKR and VOR) together with head stabilization (VCR). Some robotic controllers provide mechanisms involving eyes and head, but they usually consider a unique kinematic chain (neck-eyes) with the aim of stabilize the image on the retina (Roncone et al., 2014; Habra and Ronsse, 2016) without any learning mechanism involved. With the aim of providing an efficient regulation of the gaze system, in this work a complete model of gaze stabilization based on the coordination of VCR and OKR/VOR is presented. By inspiration on the aforementioned cerebellar theories, an adaptive learning mechanism is integrated in the model. This model is tested on a humanoid robot on three sets of experiments. The first set of experiments focused on the controller response to a set of disturbance frequencies along the vertical plane. The second shows the performances of the system under three-dimensional disturbances. The last set of experiments was carried out to test the capability of the proposed model to stabilize the gaze in locomotion tasks.

Moreover, in addition to showing how the adaptive learning mechanism can be used to improve the performances of gaze stabilization while executing a specific task that is long enough to allow for a proper learning phase, it is shown how such mechanism can be employed to perform learning even when the trials are short or to create a more general controller capable of adapting to various execution tasks. This is implemented through an offline learning technique that allows to create training sets, upon which learning is performed, and then to store the internal model parameters after learning for later use in a modified version of the stabilizing controller. This also allows the proposed method to be used in short, but repeatable (up to a certain degree of similarity) task, such as biped locomotion.

This is the first time that such a comprehensive control model, with its offline learning capability, is employed for the gaze stabilization on a physical humanoid robot.

3.2 Gaze Stabilization controller

The VOR-OKR-VCR stabilization system comprises feedforward and feedback controllers and learning networks that provide corrective control signals through internal models. The overall architecture of the controller is shown in Figure 3.1.

3.2.1 Internal model

The implementation of the internal model aims at replicating the functionalities of the cerebellum and was proposed in (Tolu, Vanegas, Luque, et al., 2012; Tolu, Vanegas, Garrido, et al., 2013). It comprises a machine learning algorithm, Locally Weighted Projection Regression (LWPR) (Vijayakumar and Schaal, 2000), that models the cerebellar granular layer (mossy fibers and granule cells), and a linear readout modelling the integration of information performed by the Purkinje cells, denoted as Purkinje Layer (PL). Both the LWPR algorithm and the linear readout can be trained by feedback signals (climbing fibers). Given the control architecture of the subsystems, the internal models act as inverse dynamic models that take as an input the

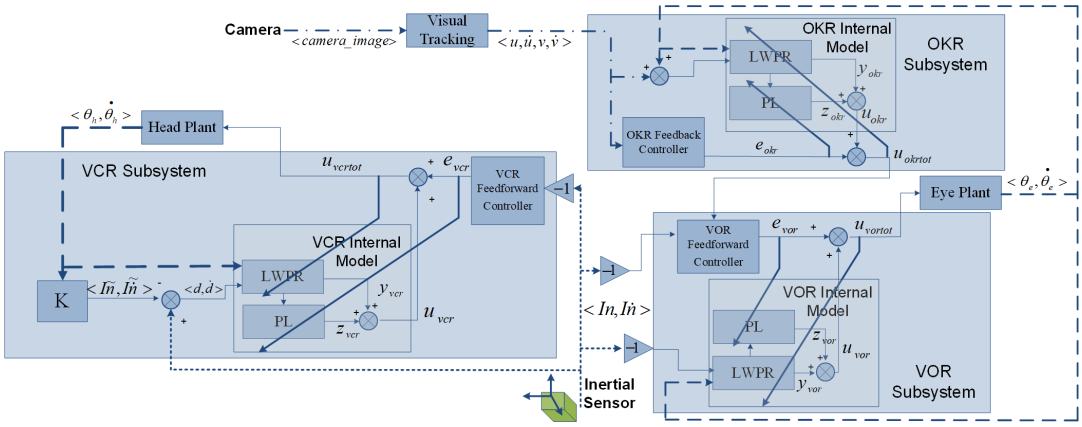


FIGURE 3.1: The proposed Gaze Stabilization controller. All three subsystems consist of a feedforward or feedback controller and the relative internal model, whose components, LWPR and PL, contribute in the generation of the motor commands, by providing appropriate corrections. The training signals for the PL and LWPR are the output of the feedforward or feedback controller and the total motor command, respectively.

reference and the current state of the robot, and produce the desired motor command that minimize the output of the feedforward or feedback component (Tolu, Vanegas, Luque, et al., 2012). This, in turn, is dependent on both the sensory delays and the robot dynamics itself, so the internal model is able to provide motor commands that compensate for these delays, therefore improving the performances of stabilization system.

The LWPR is a non-linear regression model that divides the input space into a set of receptive fields. Each one has a centre c_k and an extension defined by a positive definite distance matrix D_k. The activation of each receptive field k in response to an input x and its output are expressed by

$$p_k(x) = exp\left(-\frac{1}{2}(x-c_k)^T D_k (x-c_k)\right), \tag{3.1}$$

$$y_k(x) = w_k \cdot x + \epsilon_k, \tag{3.2}$$

where w_k and ϵ_k are the weight vector and bias associated with the k-th linear model. With every new input, the centre and the distance matrix associated with the assigned receptive field are updated.

The global output of the LWPR is given by the weighted mean of all the outputs y_k of the linear local models created:

$$y(x) = \frac{\sum_{k=1}^{N} p_k(x) y_k(x)}{\sum_{k=1}^{N} p_k(x)}. \tag{3.3}$$

By employing localized linear models, the LWPR algorithm lowers the computational cost and enable the possibility for online, real-time learning.

The activation response of the LWPR, p_k, is used to compute the output of the Purkinje Layer:

$$z(x) = \sum_k r_k p_k(x), \tag{3.4}$$

where r is a set of weights that can be trained with an update rule derived from (Porrill and Dean, 2007):

$$\delta r_k = -\beta e(x) p_k(x), \tag{3.5}$$

where β is a positive learning rate and $e(x)$ is the error signal for the current input.

The output of the internal model is the sum between the output of the LWPR and of the PL:

$$u(x) = y(x) + z(x). \tag{3.6}$$

3.2.2 Vestibulocollic reflex subsystem

In this subsystem, the feedforward controller (e_{vcr}) provides velocity motor commands that stabilize the head against disturbances originated by torso movements. The output of the internal model (u_{vcr}) is added to the motor command in order to provide corrective sensory-motor signals, thus increasing the performances of the system. The VCR Feedforward Controller is implemented as a PD controller, and its output is computed as a function of the inertial readings (orientation as Euler angles, In, and rotational velocities, $\dot{In}$):

$$e_{vcr} = k_p \cdot (-In) - k_d \cdot (-\dot{In}). \tag{3.7}$$

Albeit the choice of using a PD controller to model the Feedforward component of the reflexes is not biologically inspired, the aim of this work is a robotic application, therefore such simplification can still be considered appropriate in this context. The internal model receives as input the current and the desired angular position and velocity of the robot head, and it is trained with the output of the feedforward controller and the resulting motor command. The desired angular position and speed can be computed as the current intensity of the external disturbance. This can be estimated using readings coming from the inertial measurement unit and the encoder values, using only direct kinematics functions. The disturbance vector can be computed as $d = In - \tilde{In}$, i.e. by subtracting the expected angular rotations given by the encoder values ($\tilde{In}$) from the inertial readings (In). $\tilde{In} = [\varphi, \vartheta, \psi]$ are the Euler angles for the rigid roto-translation matrix $K(\theta_h)$ from the root reference frame to the inertial frame, computed as:

$$\varphi = atan2(-K(\theta_h)_{2,1}, K(\theta_h)_{2,2}), \tag{3.8}$$

$$\vartheta = asin(K(\theta_h)_{2,0}), \tag{3.9}$$

$$\psi = atan2(-K(\theta_h)_{1,0}, K(\theta_h)_{0,0}). \tag{3.10}$$

The angular velocity of the disturbance can be estimated as follows:

$$\dot{d} = \dot{In} - \dot{\tilde{In}} = \dot{In} - J(\theta_h) \cdot \dot{\theta_h}, \tag{3.11}$$

where J is the geometric Jacobian from the root reference frame to the inertial frame.

3.2.3 Vestibulo-ocular reflex subsystem

The VOR subsystem receives in input the head angular position and velocity signals, acquired through the inertial measurement unit ($In, \dot{In}$), and use them to generate appropriate velocity motor commands that compensate for head motion. As for the VCR subsystem, it comprises of a feedforward controller, implemented as a PD, and

an internal model. The output of the PD controller can be computed as

$$e_{vor} = k_p \cdot (-In) + k_d \cdot (-\dot{In}) + u_{okrtot}, \tag{3.12}$$

where u_{okrtot} is the output of the OKR subsystem, employed as compensatory signal (Shibata, Vijayakumar, et al., 2001). The VOR internal model receives as inputs the desired eye rotations (i.e. inertial angular positions and velocities), along with the current eye encoder values (θ_e and $\dot{\theta}_e$). As in the VCR subsystem, the learning signals are the output of the feedforward controller and the generated motor command.

3.2.4 Opto-kinetic reflex subsystem

The position error of the visual target, computed as its angular position on the horizontal and vertical axes in the camera reference frame (u, v), and the retinal slip, its derivative $(\dot{u}, \dot{v})$ are used by the OKR subsystem to keep the camera image stable. The angular position is extracted from the camera image using colour filtering, centroid computation and a conversion from pixels to angles. This information is used to compute the output of the feedback controller:

$$e_{okr} = k_p \cdot (u, v) + k_d \cdot (\dot{u}, \dot{v}). \tag{3.13}$$

As in the previous cases, the learning signals for the OKR internal model are the output of the feedback controller and the generated motor command. The inputs to the model are the current eye state and the sum of the eye position and velocity and the tracked target position and velocity, that are the absolute value of the camera image motion in the eye reference frame.

3.2.5 Offline learning

While the proposed model is capable of online, real-time learning, this is not suitable for every situation. In fact, once the internal model has been learnt, it could be used on subsequent trials in order to avoid the training phase. Moreover, the training could be performed offline, either by collecting a set of trials that can be used as a training set or by generating a random disturbance. The first approach can be useful to train the controller for a specific task which is too short to allow online learning, while the second one can be employed to expose the controller to a wider range of disturbances, creating a more generic, adaptive controller which can be later employed under different disturbances. Once the learning phase is over, the LWPR and PL parameters can be stored and used in a new trial in order to avoid having to perform the learning again. In particular, the parameters that have to be saved for the LWPR are the number of receptive fields, the distance matrices D_k, the weight vectors w_k and the biases ϵ_k, while for the PL the vector of weights r must be saved.

After the learning phase, the contribution of the non internal model term (feedforward or feedback controller) to the resulting motor command tends to be null (Tolu, Vanegas, Luque, et al., 2012). Thus, in the offline learning case, the feedback controllers for the subsystems can be not used simplifying the control scheme as shown in Figure 3.2.

3.3 Experimental setup

In order to assess the performances of the proposed model, four different kinds of experiments have been performed:

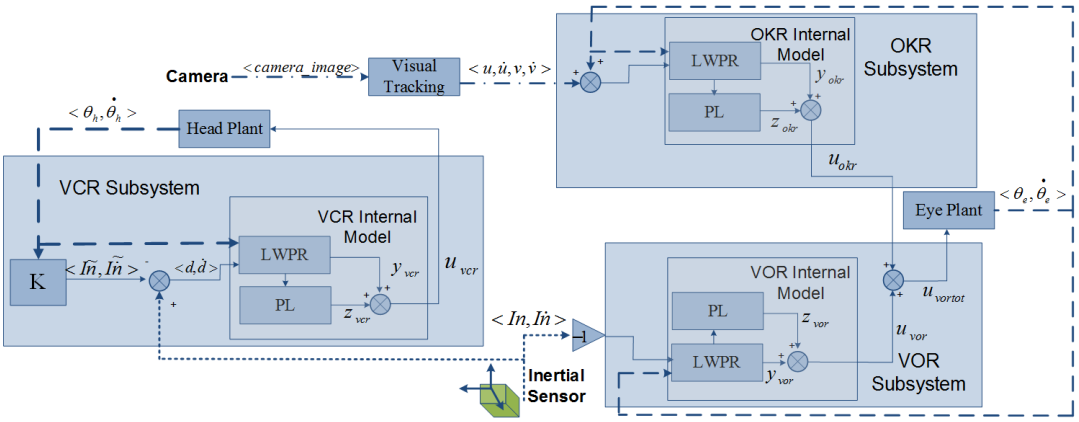


FIGURE 3.2: The modified Gaze Stabilization for the execution after a learning phase.

- stabilization of artificially generated disturbances;
- stabilization of 3D disturbances;
- stabilization of locomotion disturbance;
- stabilization of human inertial data.

In the first case, the iCub robotic head was mounted on a oscillating platform capable of generating disturbances through periodic motion (see Figure 3.3a). The Inertial Measurement Unit (IMU), mounted in the centre of the head, is an XSens MTx unit. Inertial measurements, with an angular accuracy on the roll and pitch axis of 2 degrees can be sampled with a 100Hz frequency, while the dragonfly 2 camera was set to send images with a resolution of 320x240 pixels and at frame rate of 60Hz.

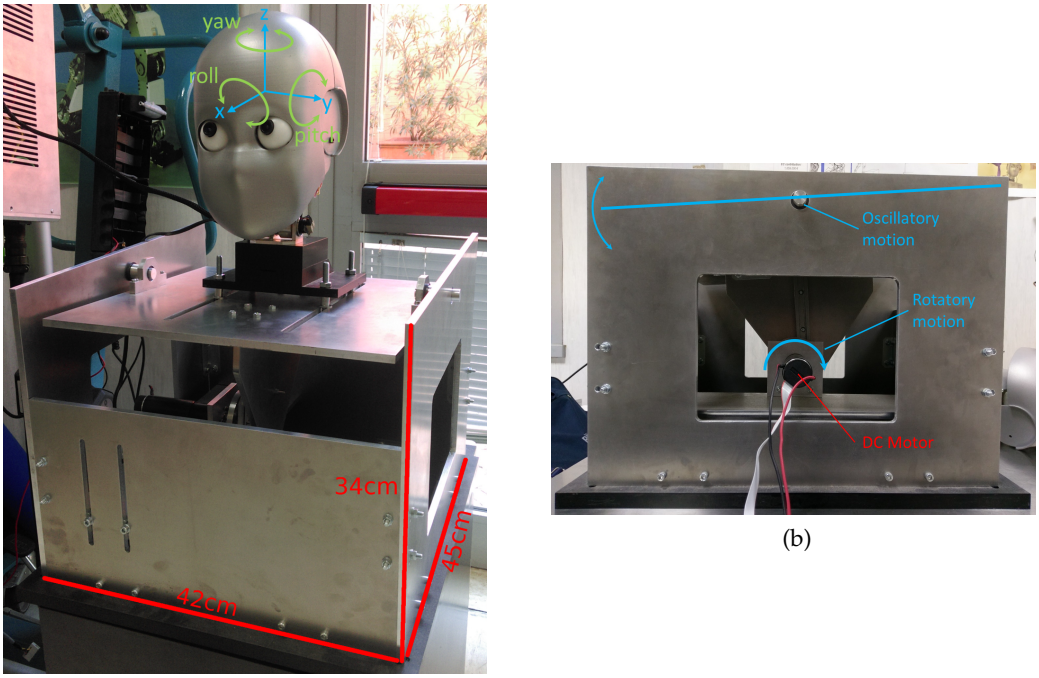
The oscillating platform is a custom built device that generates periodic disturbances and it consists in an aluminium plate connected to a rigid external frame by two coaxial bearings. The device to be tested is placed on the plate and anchored mechanically. The plate is then connected to a rigid mechanism that transforms the continuous rotation of a crank, actuated by a DC motor, through a slide mechanism, to the periodical oscillation of the plate (Figure 3.3b). The platform can only provide disturbances on a single rotational axis, and the rotation of the plate (γ) can be expressed as a function of the crank angle (δ):

$$\sin(\gamma) = \frac{R \cdot \sin(\delta)}{\sqrt{(R \cdot \sin(\delta))^2 + (L - R \cdot \cos(\delta))^2}}, \tag{3.14}$$

where R is the distance from the motor axis and the pin joint linking the disk and the slider and L is the distance between the axis of the motor and the axis of rotation of the platform. By construction these parameters are set to $R = 2$cm and $L = 20$cm.

Due to the limitations of the platform, that provides disturbances on a single rotational axis, only the pitch axis could be considered for the experiments, thus only rotations around the y axis of the inertial reference frame.

In order to verify that the controller is capable of working also for disturbances on all rotational axes, experiments were also performed on the iCub robotic simulator (Tikhanoff et al., 2008). In this case, the disturbance is provided by moving the robot torso.



(a) (b)

FIGURE 3.3: The oscillating platform. In (a) the iCub head mounted on the platform, with its inertial reference frame is shown. The transmission of motion from the DC motor to the oscillating platform is depicted in (b).

For the stabilization of locomotion disturbance, the SABIAN humanoid robot, in a straight walking task, was employed. This robot has an iCub robotic head mounted on the body of the WABIAN-2 humanoid robot (Ogura et al., 2006), that has 7 DOF in each leg and 2 DOF in the waist, with a bio-inspired range of motion. These make the robot able to perform human-like walking, with stretched knees and raising the hip, in contrast with most humanoid robots that walk with the knees bent. As the iCub head encoders do not provide velocity information, a smooth differentiation technique was employed to compute the velocity of joints. This introduced a sensory delay of around 100ms. To align all sensory feedback, the position, inertial and visual information was also delayed accordingly.

Finally, to assess the effectiveness of the stabilization capabilities of the proposed model rejecting the torso disturbance measured in real walking tasks, human inertial data relative to locomotion tasks was collected. The experiments were conducted on a human subject with no visual and vestibular impairments. An inertial measurement unit (IMU) was placed on the back of the subject, near T10, the tenth vertebra of the thoracic spine, as depicted in Figure 3.4. The IMU used was an Xsens MTi orientation sensor[1], that incorporates an on-board sensor fusion algorithm and Kalman filtering. The inertial unit is able to produce the current orientation of the torso at a frequency of 100Hz. Three different tasks were performed by the subject: straight walking (25m), circular walking and straight running (25m). The circular walking was carried out by asking the subject to walk with a circular pattern, without any indication of the pattern on the ground. Such task was executed both on normal and soft ground, provided by placing a foam rubber sheet on the ground. The foam had

[1] https://www.xsens.com/products/mti/

a density of 40kg/m^3 and the sheet measured 103x160x15cm. All tasks were performed with bare feet. Due to the fact that the inertial readings relative to the yaw rotational axis (rotation around z) can often be inaccurate because of drifting, it was decided not to use such readings. Moreovoer, in order to prevent drifts of the sensor measurements on the other two rotational axis (pitch and roll, rotations around y and x respectively), each trial lasted less than one minute with a reset of the rotational angle at the beginning of the trial (Bergamini et al., 2014). The recorded data was then employed to move the torso of the iCub robot in simulation, reproducing human locomotion.

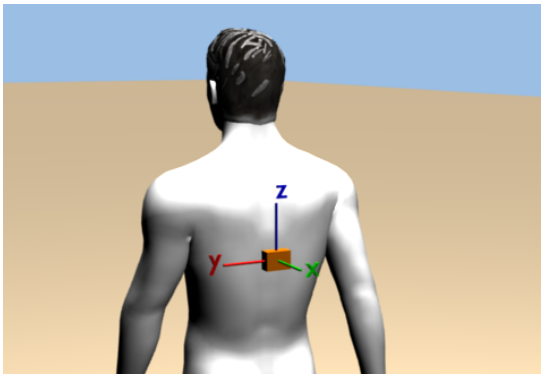


FIGURE 3.4: Placement of the inertial measurement unit on the subject.

3.4 Results

During all the experiment the main measure of error considered is the movement of the camera image (retinal slip). Human vision is considered stable if such error is below 4deg/s (Collewijn, Martins, and Steinman, 1981). In order to compute the retinal slip, and to generate the input for the OKR subsystem, a target was placed in the field of view of the robot camera. The other measure of performance considered is the inertial orientation of the head, read through the IMU.

3.4.1 Stabilization on the oscillating platform

During these trials, the robotic head was mounted on the oscillating platform that was controlled in order to produce different disturbance motions. In particular the frequency of the periodic disturbance (f) was varied during the tests.

In a first set of trials, due to their length and repeatability, no information about the learnt internal models was retained, thus the online learning phase of the LWPR and the linear readout always started from scratch at the beginning of the trial. During these tests, the frequency of motion was changed in an increasing fashion, thus providing trials of increasingly disturbance speed. During these trials, the target was kept static in front of the platform. Due to the target being static, the OKR subsystem does not provide substantial contributions during these trials, thus it was disabled. This also improves learning times, as the OKR output can slow down the learning, especially at high frequencies. Results for f ranging from 0.3Hz up to 1.5Hz can be found in Table 3.1 where, for each frequency, the Root Mean Square (RMS) value of the disturbance speed $\dot{d}_p$ (deg/s), inertial sensor readings, In_p (deg) and $\dot{In}_p$ (deg/s), and the position and velocity of the target on the camera image on the vertical axis,

v (deg) and $\dot{v}$ (deg/s), are presented. In order to avoid having the results influenced by the instability of the initial learning phase, the RMSs were computed only on the last 5 of the 14 total seconds of the trials. It can be observed that the controller is able to stabilize the image on the camera against disturbances generated by motion frequencies up to 1Hz. After that, the retinal slip ($\dot{v}$) increases over 4deg/s. However, by comparing the magnitude of the disturbance and the retinal slip, it emerges that the controller is able to reduce it to one third even for frequencies up to 1.4Hz, thus providing a substantial contribution towards a stable camera image. An example of such trials (for f = 1Hz) can be seen in Figure 3.5. It can be observed that, after an unstable learning phase at the beginning, the magnitude of the error is reduced dramatically over time, in both the velocity and position spaces. At the same time, the control signal coming from the internal model, especially from the LWPR component, increases, while the output of the feedforward controller (FFC) is reduced towards zero. This implies a switch to a sensory-motor compensatory motion that is able to almost nullify the disturbance.

TABLE 3.1: Results for disturbances provided by the oscillating platform at various frequencies.

f	$\dot{d}_p$	In_p	$\dot{In}_p$	v	$\dot{v}$
0.30	9.46	0.32	1.16	0.29	0.94
0.50	15.79	0.17	1.40	0.30	1.06
0.70	18.58	0.18	1.78	0.34	1.01
0.85	22.54	0.32	3.33	0.48	0.99
1.00	25.66	0.78	7.31	1.15	2.20
1.20	29.62	0.78	11.17	2.61	7.94
1.40	34.98	1.21	15.74	2.06	9.44
1.50	39.70	2.59	28.68	3.55	27.16

It is also relevant to stress the importance of the combination of VCR and VOR subsystems during these tasks: in fact, as shown in Table 3.1, while the VCR subsystem is able to provide a substantial contribution towards stabilization, reflected in the low inertial readings RMS values, it is only thanks to the added contribution of the VOR subsystem that full stabilization is achieved in some cases. Moreover, if employed separately, the VCR and VOR subsystems cannot achieve the same performances, as depicted in Figure 3.6, where the results for trials where only one of the two subsystems was active are shown. It can be observed that each of the two subsystems work and is able to reduce the error generated by the disturbance. However, the RMS value for the retinal slip is respectively 7.86deg/s and 4.01deg/s for the VOR and VCR subsystems. Thus, only with the combination of the two, a stable camera image is ensured.

In order to simulate the drifting motion that elicits the OKR response, the target, placed in front of the platform, was moved with a sinusoidal motion on the vertical axis with a frequency of 0.15Hz and a peak-to-peak amplitude of 15deg in the camera reference frame, while the frequency of the disturbance was kept at 1.0Hz. The behaviour of the system during this trial is shown in Figure 3.7. It can be noticed that the controller is able to stabilize the image on the camera and almost no trace of the target movement can be observed in the position or velocity space plots. The RMS value of the retinal slip in this trial is 3.69deg/s, thus the camera image is kept stable, even if the error is increased, compared to the trial with a static target.

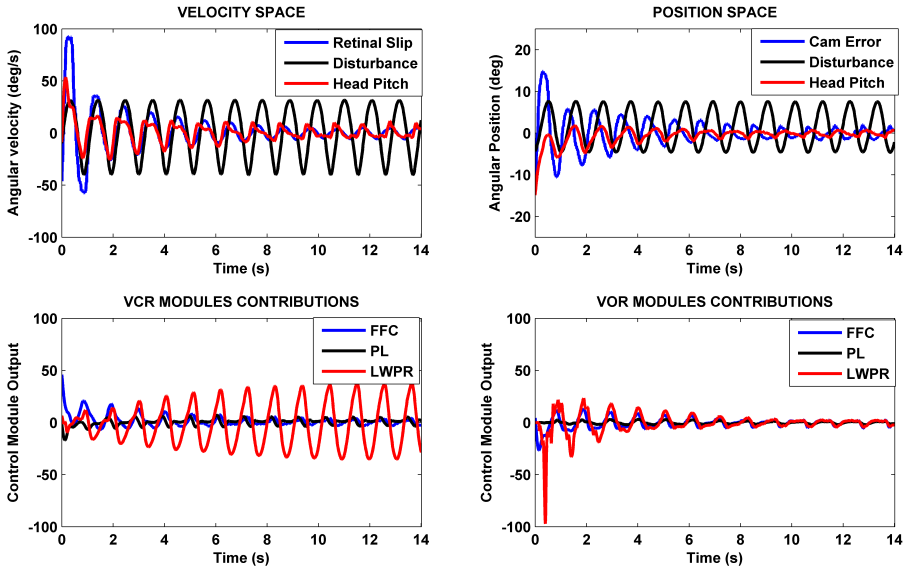


FIGURE 3.5: Execution trial for f = 1Hz. Results for the inertial readings and camera error in the velocity and position spaces are on the upper half, while the contributions of the feedforward controllers (FFC) and the internal model components (LWPR and PL), for both the VCR and VOR, are on the lower half.

In a second set of tests, the offline learning strategy was employed to create a more general controller that can work on different disturbances. To achieve this, a training phase of 150s was performed. During this phase, the platform was moved with a sinusoidal signal whose frequency randomly varied over time. The motion was generated by varying the voltage applied to the motor using a colored noise signal (low-pass filtered white noise) with a maximum voltage of 3V. This led to frequencies of motion up to 1.2Hz. Figure 3.8 shows the behaviour of this learning phase. It can be observed that, after an initial phase of circa 50 seconds where the learning occurs, the RMS of the inertial and camera measures, averaged over time bins of 3s, remain stable and no further improvements are noticeable. By performing a similar trial with the learnt internal models (Figure 3.8, bottom row), we can observe that initial unstable phase has disappeared and that the disturbance is immediately compensated. After learning, the controller was employed with the learnt internal models on a test trial in which a sinusoidal signal whose frequency gradually increased every 5 seconds was applied to the platform. Figure 3.9 shows the behaviour of the RMS during such trial, averaged over time bins of 1s. Being already trained, the system is able to immediately stabilize the disturbance and the performances do not significantly decrease when the signal changes frequency. By comparing the behaviour of the same task in the online learning scenario, it can be noticed that in the latter case the performances are inferior as the system needs to learn after each frequency change and it has not enough time to properly do so. Therefore, a long offline training phase provides a viable solution to create more general controllers capable of adaptation to different tasks.

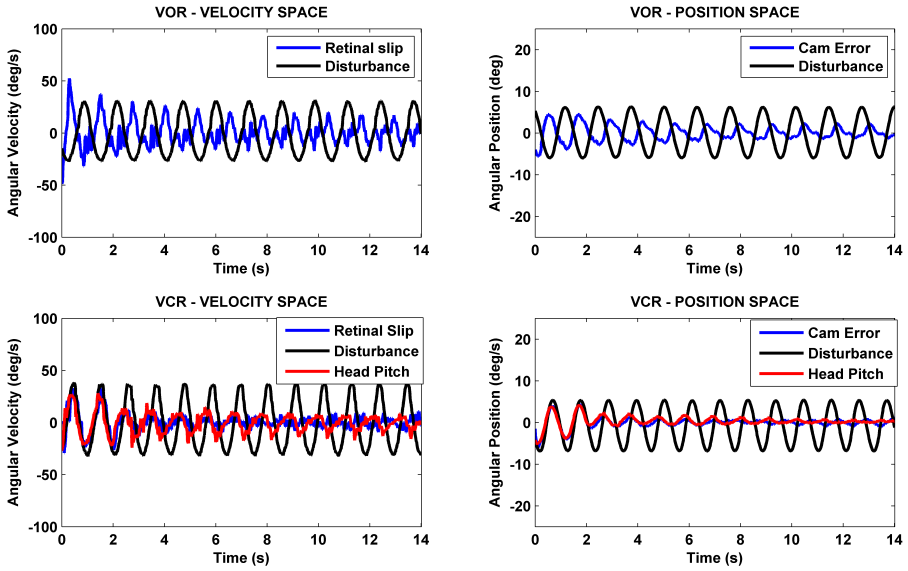


FIGURE 3.6: Execution trial for f = 1Hz with only one of VCR or VOR subsystem active.

3.4.2 Stabilization of 3D disturbances

Due to the movements of the oscillating platform being limited to only one rotational axis, stabilization of disturbances applied on all three axes could only be done in simulation.

In these tests, sinusoidal disturbances were applied to the three joints of the torso to produce a three-dimensional disturbance. In this case, the VCR is able to provide stabilizing motor commands on all three rotational axes, while VOR and OKR can only stabilize on the pitch and yaw rotational axes, as the robot eye joints do not rotate around the roll axis. Results for f ranging from 0.3Hz up to 1.5Hz can be found in Table 3.2, where the same terminology of Table 3.1 is used and r, y, u are the roll axis, yaw axis and horizontal axis of the camera image, respectively. It can observed that, while the inertial reference frame is stabilized by the VCR, the image on the camera appear to be not stable. This is because the combination of all the three rotational movements produces a linear motion of the head in space, whose effects cannot be canceled by the controller, as linear measurements are not taken into account as inputs. This motion is particularly evident on the horizontal axis. This disturbance can only be partially canceled by the OKR, especially at high frequencies.

An example of trial, for f = 1Hz, can be seen in Figure 3.10, where the disturbances, as well as the errors, on all rotational axes are shown. In accordance to the results observed for the oscillating platform, after an initial unstable phase, where learning occurs, both the inertial measurements and the camera errors decrease. While full stability of the camera image cannot be achieved due to the translation of the head caused by the torso movements, the disturbance is greatly reduced by the controller.

3.4.3 Stabilization during locomotion

The effectiveness of the controller was also tested in a realistic situation by employing the biped humanoid robot SABIAN for a walking task consisting of 8 steps in a

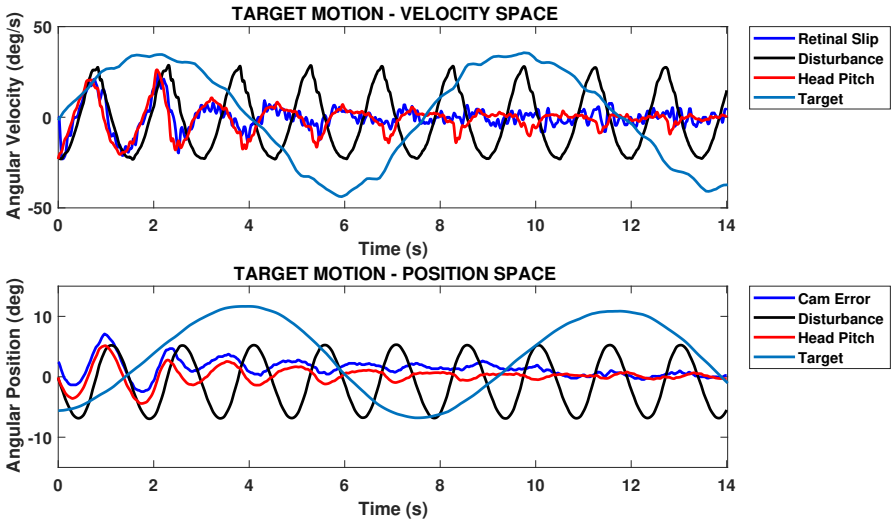


FIGURE 3.7: Execution trial for f = 1Hz with elicitation of the OKR through a vertical movement of the target.

TABLE 3.2: Results for disturbances provided by moving the robot torso on three axes, in simulation.

f	$\dot{d}_p$	In_r	$\dot{In}_r$	In_p	$\dot{In}_p$	In_y	$\dot{In}_y$	u	$\dot{u}$	v	$\dot{v}$
0.30	13.34	0.18	0.69	0.51	1.26	0.14	0.53	4.83	2.26	2.29	0.85
0.50	22.32	0.31	1.80	0.39	1.94	0.13	0.53	4.83	3.81	2.29	1.59
0.70	31.10	0.30	1.99	0.23	1.66	0.13	0.95	4.79	5.30	2.30	2.60
0.85	37.69	0.30	2.25	0.51	3.13	0.17	1.35	4.80	7.56	2.27	3.22
1.00	44.26	0.60	3.90	0.31	2.71	0.03	0.54	4.84	8.13	2.31	4.31
1.20	52.83	0.13	1.96	0.42	3.89	0.09	1.92	4.84	11.21	2.29	4.59
1.40	61.40	0.12	2.19	0.13	2.44	0.17	2.70	4.84	15.24	2.29	5.83
1.50	65.52	0.53	5.95	0.56	6.22	0.30	4.32	4.83	17.54	2.30	6.18

straight line (the step length was 160mm, step width 120mm, step time 1s). The walking control system of the WABIAN/SABIAN is composed by two modules phases (H.-O. Lim, Y. Yamamoto, and Takanishi, 2002; Yamaguchi et al., 1999; H.-O. Lim, Kaneshima, and Takanishi, 2002). The first module uses a pattern generator that calculates the trajectory of the end-effectors (feet) and generates an ideal Zero Moment Point (ZMP) and the motion pattern of the lower limbs before the beginning of the walking task. The second module is active during the walking task, for the dynamic balance of the robot, providing corrections for leg and waist motions (Torso Position Control) according to the difference between the current and the ideal value of the ZMP positions (H.-O. Lim, Ogura, and Takanishi, 2006; Yamaguchi et al., 1999; Hashimoto et al., 2012; Kang et al., 2012). In order to compute the camera error and the retinal slip, a target was placed in front of the robot, at the same height as the robot head. The target was kept static during these trials, thus the OKR module have been disabled. During a walking task, oscillations are produced on all three rotational axes, thus a stabilizing mechanism is needed for pitch, roll and yaw disturbances. However, due to the drifting of the IMU readings, it was not possible to perform stabilization on the yaw rotational axis. Therefore, the VCR was

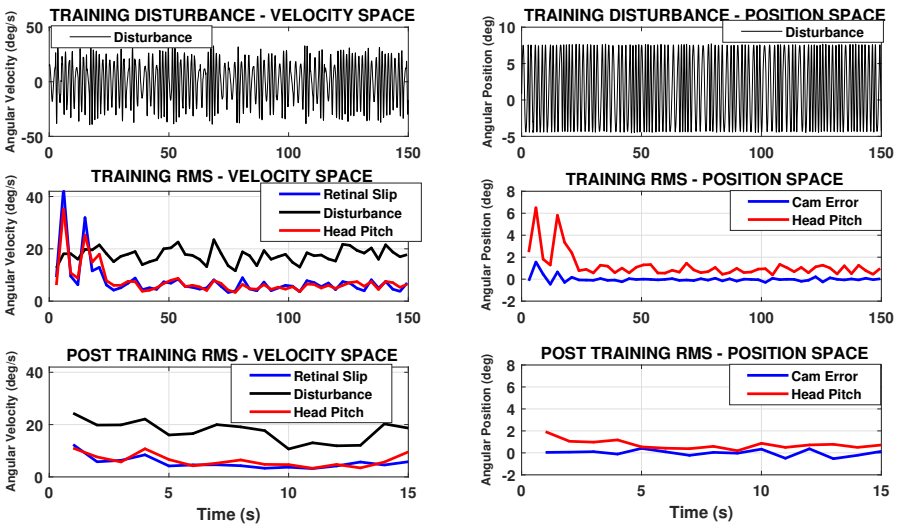


FIGURE 3.8: Offline learning with a randomly generated movement applied to the oscillating platform. The disturbance used for the training is shown on the top, the trend of the RMS of the disturbance and error signals during the training phase is show in the middle, while the behaviour of a similar trial, performed after training, is shown on the bottom.

only employed on the pitch and roll axes. Moreover, the robot eyes cannot rotate around the roll axis, thus VOR compensation is provided only on the pitch axis.

Due to the shortness of the execution, each step lasts around one second for a total of 8 seconds, it was impossible for the internal models to have enough time to properly learn the motion dynamics. In addition, the disturbance in the velocity space had some considerable peaks probably due to the impulsive acceleration the head is subject to (see not stabilized gaze in Figure 3.11), that could slow down the learning. These two reasons make the online learning disadvantageous for such a short task. Thus, the offline learning strategy was employed by creating a training set consisting in inertial data recorded during 4 locomotion trials. All the collected data were glued together to create a new offline task lasting 32 seconds. Then, the controller was executed on such trial and the learnt internal models were saved. Finally, the modified version of the controller for the execution phase was employed on a new walking task. A comparison between a stabilized walk and a non stabilized one can be seen in Figure 3.11. When the stabilization is employed the camera and inertial errors are significantly reduced in both the velocity and position spaces. Table 3.3 presents the results for two kinds of tests. In the first one the Torso Position Control (TPC) was active (as for the the trials used for the learning). In the other task the TPC was disabled producing a stronger disturbance on the head, but the gaze stabilization controller was able to effectively reject the disturbance also in this case. In particular, the RMS values for the camera errors, presented in Table 3.3 (In_r and $\dot{In}_r$ denote the inertial angular position and velocity along the roll axis), indicate a significant reduction of error while the controller is employed. In particular, the retinal slip value is lower than 4deg/s thus demonstrating the effectiveness of the proposed stabilization mechanism on a realistic task.

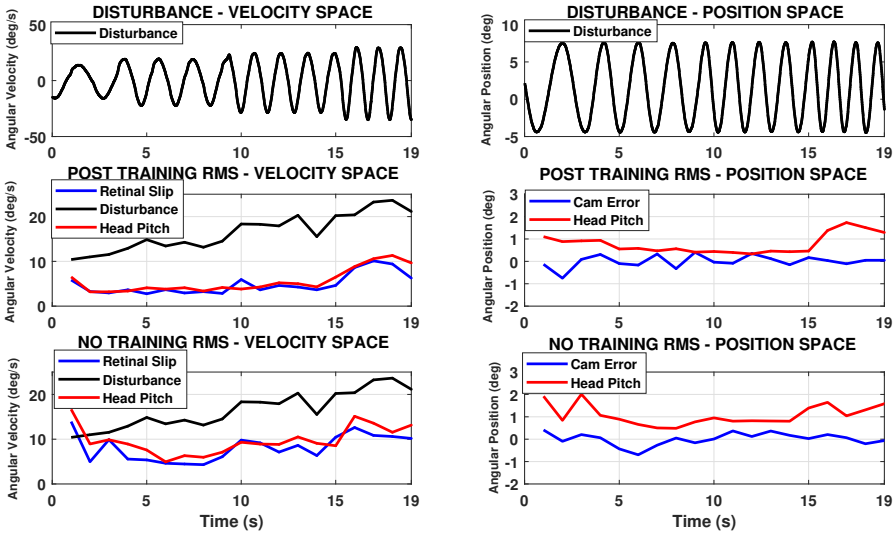


FIGURE 3.9: Comparison of the execution of the same trial by a controller trained with offline learning and another performing online learning. The frequency of the disturbance increases every 5 seconds.

TABLE 3.3: Comparison between camera and inertial RMS values in the stabilized and non-stabilized cases, in two different trials (with active and non-active TPC control).

tr.	stab.	TPC	In_p	$\dot{In}_p$	In_r	$\dot{In}_r$	v	$\dot{v}$
1	yes	yes	0.19	1.62	0.33	1.46	0.29	3.23
	no	yes	0.47	3.06	0.61	2.59	0.76	6.58
2	yes	no	0.38	1.68	0.88	2.84	0.27	3.34
	no	no	1.91	5.85	0.94	4.12	0.78	8.20

3.4.4 Stabilization of human inertial data

The stabilization model was tested on the data coming from the three different locomotion tasks (straight walking, circular walking and straight running). Due to the fact that the collected inertial data related to the yaw rotational axis was not considered, the eye-head stabilization model has been simplified, so that no stabilization on the yaw axis was performed. Moreover, given that the robot eyes cannot influence stabilization on the roll rotational axis, due to the fact that only tilt and pan motors are present, only disturbance on the pitch axis was compensated by the VOR model. Finally, as no data was recorded relative to the OKR, the relative subsystem was disabled.

For each task, a comparison between the same task performed with and without the stabilization model will be presented. The values of the gains of the PD controllers were set to $k_p = 5.0, k_d = 0.1$ for the VCR model and to $k_p = 1.0, k_d = 0.1$ for the VOR model, for all trials.

Results for the compensation of the disturbance of *straight walking* inertial data can be found in Table 3.4, where the Root Mean Square (RMS) values for inertial readings and target position and speed are presented. In this and subsequent tables, $In_p, \dot{In}_p$ are the inertial readings for rotation (deg) and rotation speed (deg/s) on the pitch axis, $In_r, \dot{In}_r$ are the inertial readings for rotation (deg) and rotation speed

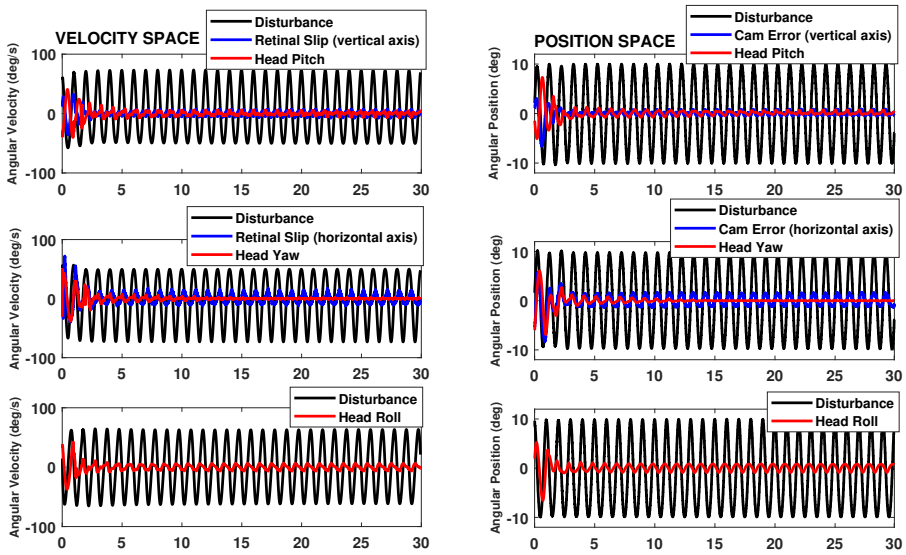


FIGURE 3.10: Stabilization of a sinusoidal motion with a frequency of 1Hz applied on all three rotational axes, in simulation. Results for pitch, head and roll are on the top, middle and bottom rows, respectively.

(deg/s) on the roll axis, $v, \dot{v}$ are the position of the target on the camera image (deg) and its speed (retinal slip, deg/s).

TABLE 3.4: Results for straight walking data.

trial	In_p	$\dot{In_p}$	In_r	$\dot{In_r}$	v	$\dot{v}$
no stabilization	2.06	11.36	2.05	11.56	3.66	8.88
stabilization	0.84	14.71	0.22	3.45	2.71	3.30

Figures 3.12, 3.13 and 3.14 show the behaviour of the task, showing the target position and retinal slip, inertial data for the pitch rotational axis and inertial data for the roll axis, respectively. From these results it can be noticed that while the roll disturbance is almost completely compensated by the VCR model, the magnitude of the rotational velocity on the pitch axis is too high to be fully compensated by the said model, that only provides an improvement in the position space. Nevertheless, the VOR subsystem is still able to maintain the camera image stable, with a mean vertical retinal slip lower than 4deg/s. Moreover, Figure 3.12 also shows a comparison between the full stabilization model and a simplified model with only the PD controllers. While the PD only implementation is able to reduce the error on the camera, it is outperformed by the complete model, thus proving the effectiveness of the latter.

Two sets of data were collected for *circular walking* tasks: one for normal ground and one for soft ground. Results for both cases are presented in Table 3.5. As in the straight walking case, the disturbance on the pitch axis cannot be fully compensated by the VCR alone, but thanks to the VOR module, the vision remains stable.

Walking on soft ground produces a greater disturbance, especially in the velocity space. Despite the higher disturbance, compared to the normal ground task, the model is still able to stabilize the head and the camera image, achieving stable vision. The behaviour of this task can also be observed in Figure 3.15.

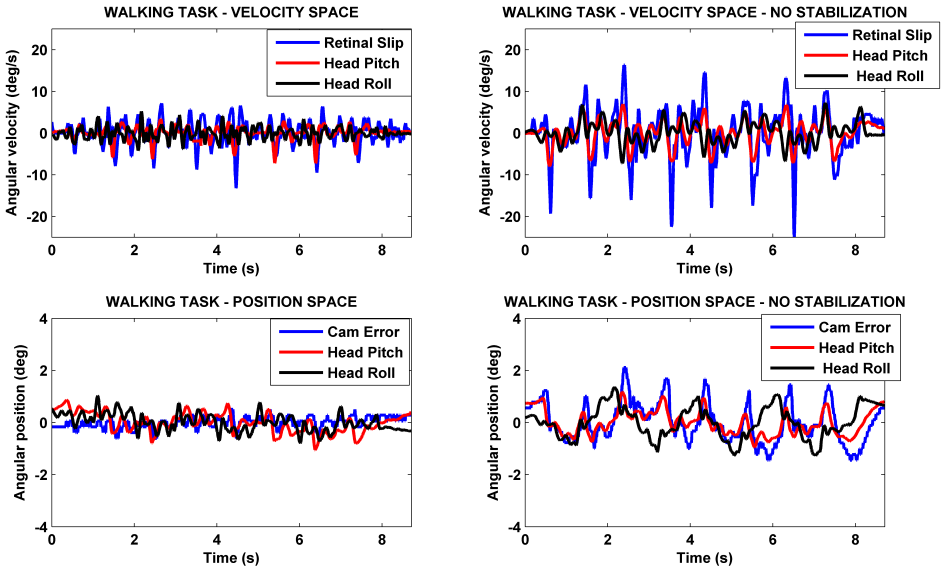


FIGURE 3.11: Stabilization during a straight walking task (with active TPC control), comparison between stabilized and non stabilized gaze.

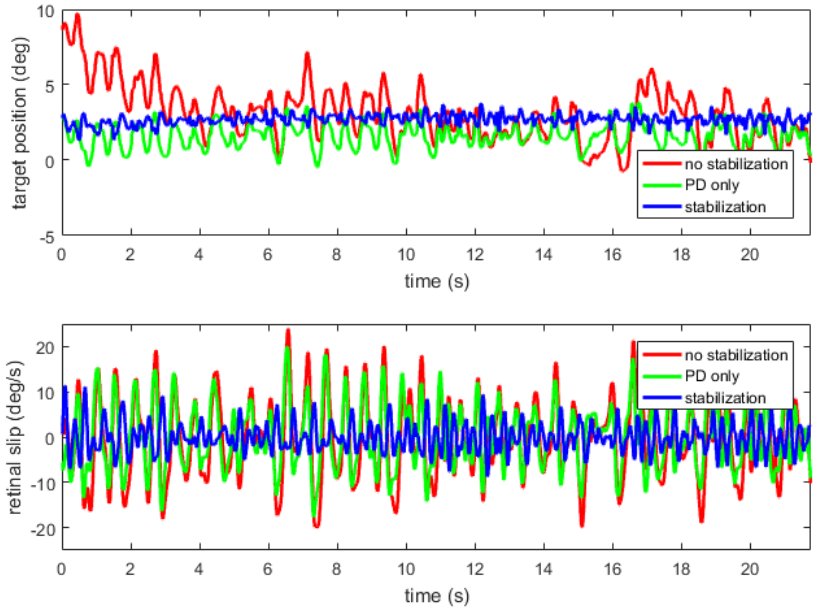


FIGURE 3.12: Stabilization task with data from a straight walking task, target position (top) and retinal slip (bottom).

During the last experiment, data from the *straight running* was used to move the robot torso. From Table 3.6 it can be observed that the model is not able to achieve a complete compensation of the disturbance, due to the high rotational velocities on the two axes. Nevertheless, the mean retinal slip is reduced to a quarter of the one of the trial with no stabilization. Thus, the model provides a viable solution even for disturbances of this magnitude, as it is also shown in Figure 3.16.

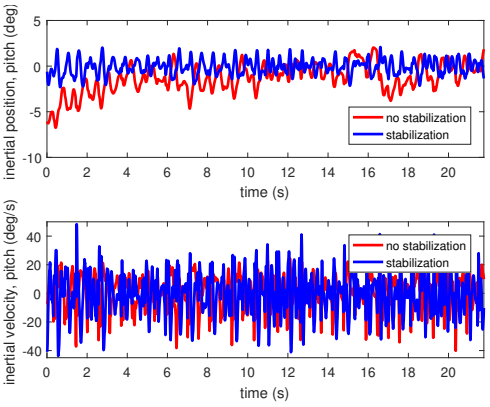


FIGURE 3.13: Stabilization task with data from a straight walking task, inertial position data (top) and inertial velocity (bottom), pitch axis.

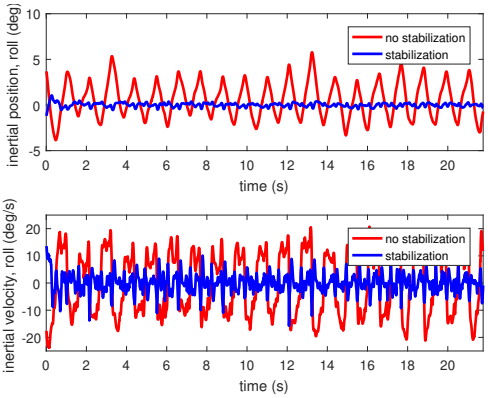


FIGURE 3.14: Stabilization task with data from a straight walking task, inertial position data (top) and inertial velocity (bottom), roll axis.

TABLE 3.5: Results for circular walking data on normal and soft ground.

ground	trial	In_p	$\dot{In}_p$	In_r	$\dot{In}_r$	v	$\dot{v}$
normal	no stabilization	3.35	14.03	2.11	6.36	3.61	12.86
normal	stabilization	0.49	6.34	0.17	2.84	0.88	1.85
soft	no stabilization	3.09	16.00	3.89	10.95	4.23	15.15
soft	stabilization	0.46	6.35	0.24	3.07	2.74	3.56

TABLE 3.6: Results for straight running data.

trial	In_p	$\dot{In}_p$	In_r	$\dot{In}_r$	v	$\dot{v}$
no stabilization	3.14	45.36	1.77	16.28	3.49	25.96
stabilization	1.06	43.03	0.53	10.78	1.61	6.36

3.5 Conclusions

In this work a complete neuro-controller for the stabilization of the gaze that is based on the coordination of VCR, VOR and OKR in conjunction with cerebellar internal models, implemented through machine learning approaches, was presented and validated through an implementation on a humanoid robotic platform. The model was tested on a robotic head mounted on an oscillating platform capable of generating periodic disturbances and on a complete humanoid robot during a locomotion task. Results on the oscillating platform show that the model is able to keep the camera image stable while the disturbance is applied. The coordination of the VCR and VOR components proves beneficial for the overall performances and the controller

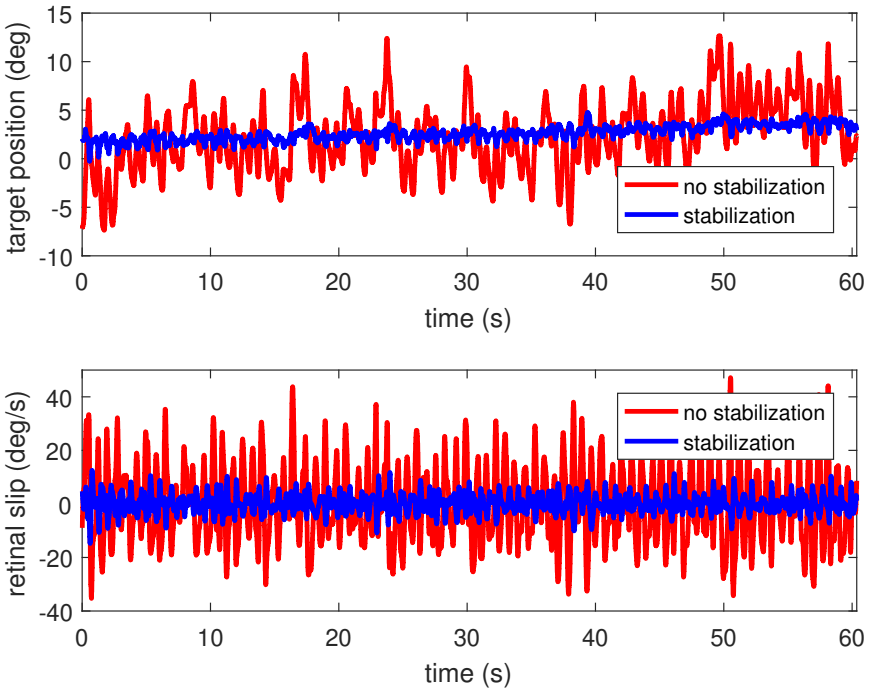


FIGURE 3.15: Stabilization task with data from a circular walking task on soft ground, target position (top) and retinal slip (bottom).

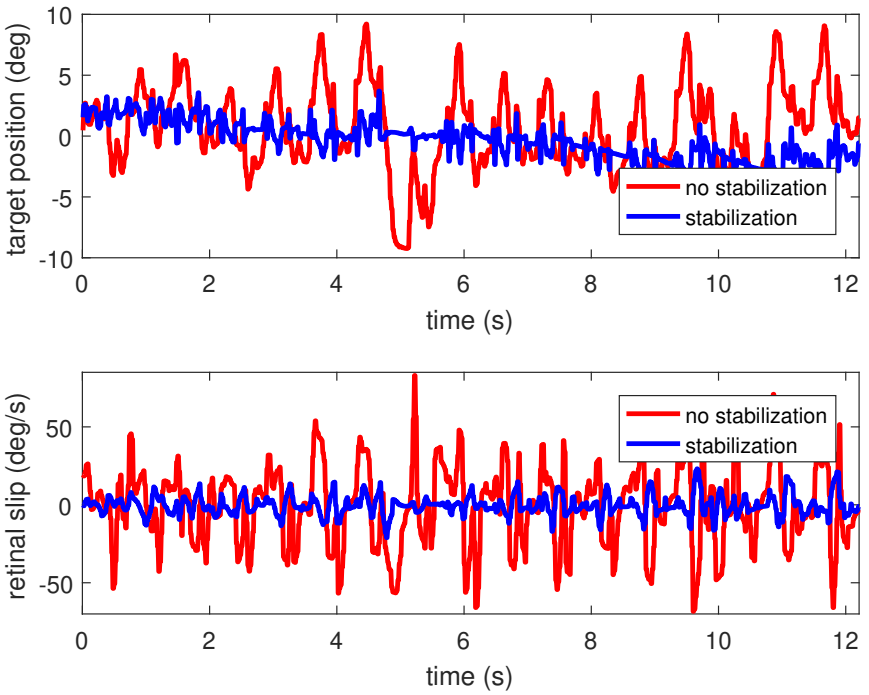


FIGURE 3.16: Stabilization task with data from a straight running task, target position (top) and retinal slip (bottom).

is able to cope with drifts of the camera image thanks to the OKR subsystem. Moreover, tests performed during robotic locomotion trials prove the effectiveness of the approach in realistic scenarios. Experiments also shown that the internal models do not have to be learnt online during the execution of the task but can be generated by employing offline training and then stored to be used on real task. Finally, results

also show that the model is able to compensate for disturbances recorded on human subject locomoting, and that the system performs well in almost all trials, with the exception of the straight running task, albeit being able to improve the stabilization by reducing the retinal slip to a quarter of the one from the task were no stabilization was present. The model still has some limitations, and could be improved in some aspects. Due to the fact that only rotational information is used by the controller, disturbance produced by linear translations cannot be compensated. However, the controller could be in principle be extended to take also these inputs into account. Another simplification introduced was to align all sensory feedback in time introducing a fixed delay. In a more biologically accurate system, the delays could be different for every sensory pathway and could also be non constant. Enabling the OKR in some circumstances can significantly increase the learning times, therefore it should be only activated if its contribution could really be helpful in stabilizing the camera image. A possible improvement in this direction could be an automatic switching mechanism that enable the OKR when its contribution is required. Overall, this work has proven that the coordination of the three aforementioned reflex can increase the performances of gaze stabilization on physical humanoid robotic platforms and than can provide a fundamental component for visually-guided locomotion.

Chapter 4

A framework for connecting robots and neural simulations

This chapter reports on the development of a software tool that enable the synchronization and data exchange between physical and neural simulations as well as providing the necessary graphic user interfaces to develop and run experiments. This tool is the Neurorobotics platform developed as part of the Human Brain Project SP 10 - Neurorobotics.

Clearly, the development of the Neurorobotics Platform was a very collaborative effort, and the personal contribution was limited to the design and implementation of the Closed Loop Engine synchronization mechanism and of the appropriate interfaces towards the robotics side and physical simulations as well as to the design and implementation of the experiments presented. However, for the sake of completeness, also other parts of the platform will be described.

4.1 Introduction

Developing neuro-inspired computing paradigms that mimic nervous system functions is a well-established field of research that fosters our understanding of the human brain. The brain is a complex structure and designing models that can mimic such a structure is particularly difficult. Modelling brain function requires understanding how each subsystem (sensory, motor, emotional, etc.) works, how these subsystems interact with each other, and, as a whole, how they can generate complex behaviours in the interaction with the environment. Moreover it is well known that during development the brain is moulded by experience and the environment (Benefiel and Greenough, 1998; Briones, Klintsova, and Greenough, 2004). Thus,

This chapter has been adapted from the articles:

Falotico, E., L. Vannucci, A. Ambrosano, U. Albanese, S. Ulbrich, J. C. Vasquez Tieck, G. Hinkel, J. Kaiser, I. Peric, O. Denninger, N. Cauli, M. Kirtay, A. Ronneau, G. Klinker, A. von Arnim, L. Guyot, D. Peppicelli, P. Martinez-Cañada, E. Ros, P. Maier, S. Weber, M. Huber, D. Plecher, F. Röhrbein, S. Deser, A. Roitberg, P. van der Smagt, R. Dillmann, P. Levi, C. Laschi, A. Knoll, and M.-O. Gewaltig (2017). "Connecting artificial brains to robots in a comprehensive simulation framework: The neurorobotics platform". In: *Frontiers in neurorobotics* 11, p. 2.

Vannucci, L., A. Ambrosano, N. Cauli, U. Albanese, E. Falotico, S. Ulbrich, L. Pfotzer, G. Hinkel, O. Denninger, D. Peppicelli, L. Guyot, A. Von Arnim, S. Deser, P. Maier, R. Dillman, G. Klinker, P. Levi, A. Knoll, M.-O. Gewaltig, and C. Laschi (2015). "A visual tracking model implemented on the iCub robot as a use case for a novel neurorobotic toolkit integrating brain and physics simulation". In: *IEEE-RAS International Conference on Humanoid Robots*. Vol. 2015-December, pp. 1179–1184.

Ambrosano, A., L. Vannucci, U. Albanese, M. Kirtay, E. Falotico, P. Martínez-Cañada, G. Hinkel, J. Kaiser, S. Ulbrich, P. Levi, C. Morillas, A. Knoll, M.-O. Gewaltig, and C. Laschi (2016). "Retina color-opponency based pursuit implemented through spiking neural networks in the neurorobotics platform". In: *Lecture Notes in Computer Science* 9793, pp. 16–27.

studying and validating models of brain function requires a proper embodiment of the brain model as well as a dynamic and rich sensory environment in which the robot-brain ensemble can be embedded and then be exposed to a realistic sensory-motor task. Since advanced brain models are too complex to be simulated in real time, the researcher is faced with a dilemma. Either the brain model is simplified until it can be simulated in real time. In this case, the brain model can be embedded in a physical robot, operating in the real world, but the complexity of the brain models that can be studied is highly limited. Or the complexity of the brain model is maintained. In this case, there are no limits on the brain models, however, it is now no longer possible to embed the brain into a real world task. Rather, the embodiment has to be simulated as well.

While, from one side, adequate tools exist to simulate either complex neural network models (Gewaltig and Diesmann, 2007) or, on the other, robots and their environments (Koenig and Howard, 2004), there is so far no tool that allows researchers to easily connect realistic brain models to a robot and embed it in a sensory rich environment model.

Such a tool would require the capability of orchestrating and synchronizing both simulations as well as managing the exchange of data between them. The goal of such simulations is to study and quantify the behaviour of models of the brain. As a consequence, we do not only need a complex, realistic experimental environment, but we also need a controllable and measurable setup where stimuli can be generated and responses can be measured. In fact, real environment complexity and parameters are intrinsically difficult or even impossible to control. In addition, models of brain functions, designed to properly reproduce brain activity at different levels could not be executed in real-time due to complex neuron dynamics and the size of the network (Kunkel et al., 2014). This is the reason why it is proposed to use a digital simulator implementing realistic scenarios. The main restriction proposed is to have a simulator that could run at a "slower" time (depending on the needed computational time of the brain) and also that the time can be sampled in discrete intervals without compromising the simulation quality.
The idea behind this approach is providing a tool chain, which grants researchers access to simulation control as well as state-of-the-art tools such as models of robot and brain and methods to connect them in a proper way (i.e. connecting spiking neural networks to robotic sensors and actuators). A first approach used to connect spiking neural networks and robots has been presented by (Gamez, Fidjeland, and Lazdins, 2012). iSpike is a C++ library that provides an interface between spiking neural network simulators and the iCub humanoid robot. It uses a biologically-inspired approach to convert the robot's sensory information into spikes that are passed to the neural network simulator, and it decodes output spikes from the network into motor signals that are sent to control the robot. Another communication interface named CLONES (Voegtlin, 2011) between a neural simulator (BRIAN, (Goodman and Brette, 2008)) and SOFA, a physics engine for biomedical applications (Allard et al., 2007), has been developed using shared memory and semaphores. The most similar system to iSpike and CLONES is the interface that was created for the CRONOS and SIMNOS robots (Gamez, Newcombe, et al., 2006) which encoded visual and proprioceptive data from the robots into spikes that were passed to a spiking neural network simulated in SpikeStream. Spiking motor output from the network was transformed back into real values that were used to control the robots. This system was used to develop a spiking neural network that controlled the eye movements of SIMNOS, learnt associations between motor output and visual input, and used models of imagination and emotion to avoid negative stimuli. All these systems provide

an interface towards specific robotic platforms able to deal with spiking/digital inputs and convert them appropriately. Together with robotic platform restrictions, they do not provide a framework for the conversion, allowing the user to write his own transfer function. A more generic system which permits dealing with simulated robotic platforms is AnimatLab (Cofer, Cymbalyuk, Reid, et al., 2010). AnimatLab currently has two different neural models that can be used. One is an abstract firing rate neuron model, and the other is a more realistic conductance based integrate-and-fire spiking neural model. It is also possible to add new neural and biomechanical models as plug-in modules. There are several different joint types, and a host of different body types that can be used. Although AnimatLab does not provide a comprehensive set of neurons and learning models, some behavior implementation based on this tool is available such as locust jumping (Cofer, Cymbalyuk, Heitler, et al., 2010) or dominant and subordinate crayfish (Issa et al., 2012). Despite some of the mentioned tools represents a good attempt to connect artificial brains to robots, these are not very common in the robotic and neuroscientific communities likely due to the limitations underlined (robotic platform restrictions, lack of a framework for conversions). For the proposed framework it was decided to rely on widely used simulators for the brain models as well as for robots and environments. This strategic choice should allow to easily attract users of these platforms. These simulators were embedded in a comprehensive framework that allows the user to design and run neurorobotic experiments. In line with the described approach, (Weidel, Duarte, et al., 2015; Weidel, Djurfeldt, et al., 2016) proposed to couple the widely used neural simulation tool NEST (Gewaltig and Diesmann, 2007) with the robot simulator Gazebo (Koenig and Howard, 2004), using the MUSIC middleware (Djurfeldt et al., 2010).

Here, the first release of the HBP Neurorobotics Platform is described. This piece of software offers scientists and technology developers a set of tools, allowing them to connect brain models to detailed simulations of robot bodies and environments and to use the resulting neurorobotic systems in *in-silico* experiments and technology development. The Neurorobotics Platform (NRP) also provides a comprehensive development framework including editors for creating experiments, environments, and brain and robot models. These tools are accessible via the web allowing them to use the platform without tedious installation of software packages. Moreover, through the web, researchers can collaborate and share their models and experiments with their colleagues or with the scientific community.

Although the capabilities to model virtual robots and environments already exist as confirmed by the mentioned works, and although various labs have created closed-loop set-ups with simple brain models (Ros et al., 2006; Denoyelle et al., 2014), this platform is the first to allow the coupling of robots and detailed models of the brain. This makes it possible to perform experiments exploring the link between low-level brain circuitry and high-level function.

The aim of this platform is twofold: from one side, the platform can be used to test neuroscientific models of brain areas, or even reconstruction of these areas based on neurophysiological data; on the other side, roboticists can take advantage of such a platform to develop more biologically inspired control architectures. The physical and neural simulation are properly synchronized and they exchange data through transfer functions that translate sensory information coming from the robot (camera image, encoders, etc.) into input for the brain (current and spikes) from one side and the network output into motor commands from the other. Additionally, the platform also provides a web interface, so that it can be easily accessed and used from a

broader user base. From this web interface the user can also access the editors that are used to construct experiments from scratch and run the experiments without any software installation, benefiting from the available computing and storage platforms that have been made available to support the NRP. Therefore, the NRP provides a complete framework for neurorobotics experiment design and simulation. One of the pillars of the NRP development is the reuse and extension of existing software, thus many components were implemented using suitably chosen existing software.

4.2 Platform Requirements

4.2.1 Functional Requirements

In order to obtain the *functional requirements* for the NRP, at first it was determined which features are needed for the creation of a neurorobotic experiment. In that, software engineering concepts and terminologies were followed to itemize platform features as requirements (IEEE, 1998). These features can be divided into two categories: *design* features and *simulation* features, each with its own functional requirements.

During the *design* of a neurorobotic experiment, the user should be able to define all of its properties and this includes:

- the design of a suitable *Robot* model, by defining both kinematic and dynamic properties as well as the appearance, either from scratch or from pre-existing models;
- the possibility to create a rich *Environment* models in which the robot can operate, by using a library of objects;
- the design of a *Brain* model, either from scratch or by selecting an existing model, that will be coupled to the robot;
- *Brain-Body Integration*, in order to specify how the brain model and the robot should be coupled in terms of sensory and motor data exchange to create a *Neurobot*;
- the capability to change dynamic properties of the *Experiment* itself, like defining events that can be triggered during the simulation and appropriate response behaviors.

When all properties are defined, the *simulation* can start. During the execution, the NRP should provide:

- *World maintenance and synchronization mechanisms* in order to not only simulate both the physics and neural models, but to synchronize the two simulations and exchange data between them, providing a closed-loop control mechanism, as defined in the design phase. It must be possible to start, pause, stop and reset the simulation. The simulation should react to the triggered events previously defined;
- a proper *Interactive visualization* of the running simulation, comprising a GUI and utilities to see details of the simulation like brain activity or robot parameters. Moreover, the user should be able to *live edit* and interact with the simulation once it is started, using the same design features described above.

An overview of the platform functionalities is shown in Figure 4.1.

4.2.2 Non-Functional Requirements

Several non-functional requirements were also defined:

- *usability and user experience* – the platform should be easily accessible to a wide range of users that possibly have no experience in either the neuroscientific or robotic fields. This should be achieved by an user-centric design with intuitive tools and a consistent user experience. Moreover, the platform should also provide an additional user level in order for expert users to have more detailed design capabilities;
- *open source* – the NRP should rely on existing building blocks, and in particular on open source ones, as the platform has to be released to a wide audience.
- *interoperability* – each software component that allows to save or load data should use, wherever possible, well-known data formats.
- *software quality* – in order to ensure software quality, the development of the platform should follow software engineering practices such as keeping a task tracking system, using version control with code review and continuous builds, and employing standard software development methodologies.

4.2.3 Integration with other HBP Platforms

The NRP is one of six platforms developed in the Human Brain Project. In addition to the Neurorobotics Platform, the HBP develops a *Neuroinformatics Platform*, a *Brain Simulation Platform*, a *High Performance and Data Analytics Platform*, a *Neuromorphic Computing Platform* and a *Medical Informatics Platform*. Most of these offer their services through the web and are built on top of a common set of APIs and services, called *HBP Collaboratory Portal*. It provides the following services:

- Authentication, access rights and user profiles. The users are provided with a Single Sign-On mechanism so they can use the same credentials to access every HBP platform.
- Document repository. The users has access to a document repository in which they can store and manage their projects. It supports one of NRP's requirement, namely the possibility for the users to share their models (brain, connections, environment, robots or experiments) with team members.
- Collaboratory API. A web based application with associated libraries allowing every platform's web interface to have the same look and feel, and to be implemented as a plugin within the Collaboratory Portal.

All the HBP platform should provide some level of integration among each other. For this reason, short-term future development plans include the integration of the Neurorobotics Platform with the Brain Simulation Platform and the Neuromorphic Computing Platform, while in the long term integration with the High Performance Computing and Analytics Platform will also be provided.

The Brain Simulation Platform aims at providing scientists with tools to reconstruct and simulate scaffold models of brain and brain tissue using data from within and outside the HBP. The Brain Simulation Platform will be integrated with the NRP for simulating brain models at various detail levels. Moreover, alongside the Brain

Simulation Platform, scaffold brain models will be gathered and they will be available for usage in the platform.

The Neuromorphic Computing Platform provides remote access to large-scale neuromorphic computing systems built in custom hardware. Compared to traditional HPC resources, the neuromorphic systems offer higher speed (real-time or accelerated simulation time) and lower energy consumption. Thus, the integration of the platform will provide an alternative neural simulation backend more suitable for simulations that require a high computational burden, such as in experiments involving plasticity and learning.

4.3 Software Architecture

The Neurorobotics Platform is based on a three layers architecture, shown in Figure 4.2.
The layers, starting from the one furthest from the user, are the following:

1. the software components simulating the neurorobotics experiment;
2. the *REST server* or *Backend*;
3. the *Experiment Simulation Viewer (ESV)*, a graphical user interface, and the Robot Designer, a standalone application for the design of physical models.

The first layer comprises all the software components which are needed to simulate a neurorobotics experiment. The *World Simulation Engine (WSE)* is responsible for simulating robots and their environment. The *Brain Simulator* is responsible to simulate the neural network that controls the robot. The *Closed Loop Engine (CLE)* implements the unique logic of each experiment and orchestrates the interaction between the two simulators and the ESV.

The second layer contains the *REST server*, also referred to as *Backend*, which receives requests from the ESV and forwards them to the appropriate components which implements the requested service, mainly through ROS. The REST server thus acts as a relay between the graphical user interface (the frontend) and the various simulation engines needed for the neurorobotics experiment. For practical reasons, the services provided by the REST server are tightly coupled with the high level functionality shown in the ESV GUI. Thus any graphical control interacting with the REST server has a corresponding service. Actions which change the state of the simulations, such starting, stopping or pausing a simulation, are implemented as a single parametric service.

The ESV is the web-based graphical user interface to all neurorobotics experiments. Using the ESV, the user can control and visualize neurorobotics experiments. The ESV also provides a number of *editors* to configure the experiment protocol as well as the parts of the experiment such as the environment, the brain model, the connection between brain and robots (*Brain Interface and Body Integrator*). The Robot Designer is a tool that was developed to allow the process of designing robot models that can be included in simulation setups executable on the NRP. This tool is developed as a plugin for the 3d modeling tool Blender 3D.

4.3.1 Brain Simulator

The goal of the *Brain Simulator* is to simulate a brain circuit, implemented with a spiking neural network (SNN).

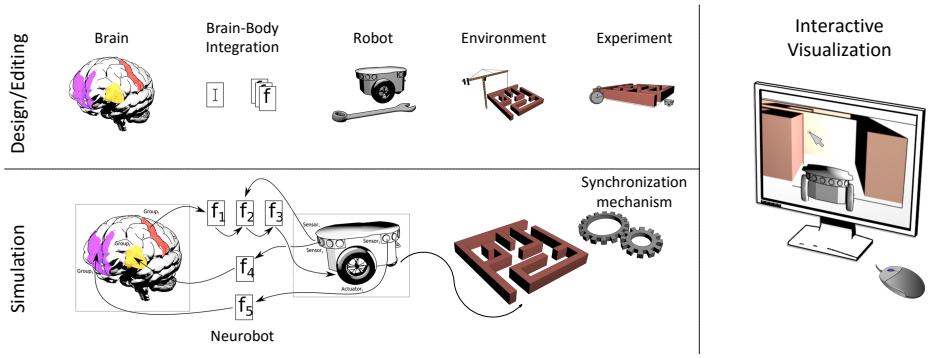


FIGURE 4.1: Functional overview of the Neurorobotics Platform. Using the design/editing features of the platform, the user is able to create a neurorobotic experiment comprising of a brain model integrated with a robotic body (Neurobot) that interacts in a dynamic environment. The experiment is then simulated by a synchronized neural-physics simulation and the results can be displayed in an interactive fashion.

Several simulators for SNNs exist, with different levels of detail, ranging from more abstract point neuron simulations, that consider neural networks as directed graphs, to the morphologically accurate ones where the properties of axons and dendrites are taken into account.

Inside the NRP, the simulator currently supported is NEST (Gewaltig and Diesmann, 2007), a point neuron simulator with the capability of running on high performance computing platforms, that is also one of the simulation backends of the Brain Simulation Platform. NEST is supported through the use of the PyNN abstraction layer (Davison et al., 2008) that provides the same interface for different simulators and also for neuromorphic processing units, i.e. dedicated hardware for the simulation of SNN such as SpiNNaker (Khan et al., 2008), provided by the Neuromorphic Computing Platform. Both NEST and PyNN provide convenient mechanisms to design neural networks. Furthermore, they are among the most used tools in the neuroscientific community. On the other hand, the only APIs they provide are written in Python, which heavily constraints the choice of the language to use for interacting with them.

4.3.2 World Simulator

In order to have realistic experiments, the accurate brain simulation must be coupled with a detailed physics simulation. The *World Simulator* component aims at delivering a realistic simulation for both the robot and the environment in which the robot interacts.

Gazebo was chosen as the physics simulator. It offers a multi-robot environment with an accurate simulation of the dynamics, in particular gravity, contact forces and friction. This dynamic simulation can be computed with different supported software libraries like ODE (Drumwright and al., 2010) and Bullet (Coumans et al., 2013).

Any communication with the simulated robot and control of the simulation itself is done through the Robot Operating System (ROS) (Quigley et al., 2009) which is natively integrated with Gazebo.

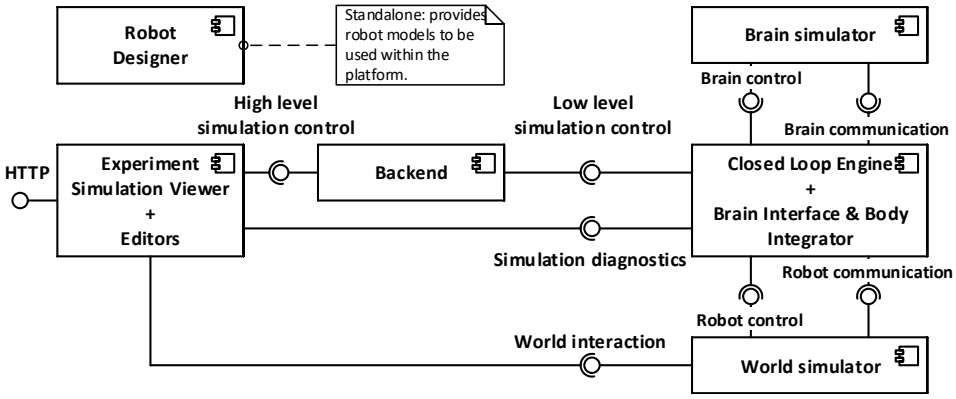


FIGURE 4.2: Architectural overview of the platform. From left to right, three layers can be distinguished: the user interface (Experiment Simulation Viewer), the services connecting the user interface to the simulations (implemented in the Backend) and the internal computations, comprising the two simulations and the synchronization between them.

ROS is a widely used middleware in the robotics community and provides C++ and Python APIs to the user.

4.3.3 Brain Interface and Body Integrator

The *Brain Interface and Body Integrator* (BIBI) plays a crucial role in the NRP, as it is the component that implements the connection between the robot and brain simulations. The main feature of the BIBI is the *Transfer Function* framework. A Transfer Function (TF) is a function that translates the output of one simulation into a suitable input for the other. Thus, two main types of transfer functions can be identified: The *Robot to Neuron* TFs translate signals coming from robot parts such as sensor readings and camera images into neuron signals such as spikes, firing rates or electric currents; the *Neuron to Robot* TFs convert neural signals from individual neurons or groups of neurons into control signals for robot motors. Thus, these two kinds of transfer functions close the action-perception loop by filling the gaps between the neural controller and the robot. This is illustrated in Figure 4.3.

The TFs also extend beyond the previously described two types. For example, the robot-brain-robot loop can be short-circuited in order to bypass the brain simulation and use only a classical robotic controller, thus resulting in a *Robot to Robot* TF. This allows the comparison between a classical and a neural implementation of a robotic controller with the same set-up, by simply switching from a transfer function to another. Moreover, the data coming from both simulations can be sent out of the loop (to a monitoring module) where it can be live plotted, elaborated, stored or exported for data analysis with external tools (*Robot to Monitor* and *Neuron to Monitor* TFs).

In order to provide a proper abstraction layer towards the simulators, generic interfaces are provided, that are then implemented by specific adapters. From the robot simulator side, the interface is modelled following the publish-subscribe design pattern (Gamma et al., 1995), where, from one side, sensory information is expected to be published by the robotic simulator and the *Robot to Neuron* TF subscribes to the subject, receiving the data, while on the other side the *Neuron to Robot*

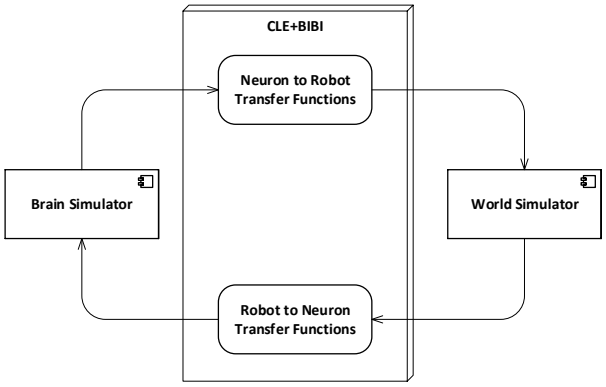


FIGURE 4.3: Closed action perception loop as performed by the Brain Interface and Body Integrator. Sensory data from the robotic simulation is transformed into spikes and current amplitudes inside a robot to neuron transfer function and sent to the neural simulation. Conversely, measurement on the spiking neural network are converted into motor commands inside a neuron to robot transfer function and sent to the robot, closing the action-perception loop.

TF publishes motor commands and the simulator is expected to subscribe and execute them. This pattern is used by many robotics middlewares such as ROS and YARP (Metta, Fitzpatrick, and Natale, 2006), thus there is minimal work required in order to implement the adapters in such cases. In the current implementation of the NRP, ROS Topic adapters have been implemented. From the brain simulation side, the TFs provide stimuli and measurements by using *Devices*. Devices are abstract entities that have to be connected to the neural network, either to a single neuron or to a neuron population. Among such entities, there are spike generators and current generators (for the input side), and spike recorders, population rates recorders, and leaky integrators (for the output side). In the current implementation, devices are implemented as wrappers around PyNN objects instances, providing general interfaces towards different neural simulators.

The TF framework is implemented using the Python programming language, where the business logic of each TF resides in a function definition. A library of commonly used transfer functions, including common image processing primitives and simple translational models for motor command generation, is provided alongside with the framework. Information about the TF connections is specified via a custom Domain Specific Language (DSL) implemented with Python decorators that specify the type of transfer function, the device types and the neuron which they are connected to, and the topics that the TF should subscribe to, or on which topic the TF should publish (Hinkel, Groenda, Vannucci, et al., 2015; Hinkel, Groenda, Krach, et al., 2016). An example of a transfer function implementation is displayed in Listing 1.

```
@nrp.MapRobotSubscriber("camera", Topic('/robot/camera', sensor_msgs.msg.Image))
@nrp.MapSpikeSource("red_left_eye", nrp.brain.sensors[0:3:2], nrp.poisson)
@nrp.MapSpikeSource("red_right_eye", nrp.brain.sensors[1:4:2], nrp.poisson)
@nrp.Robot2Neuron()
def eye_sensor_transmit(t, camera, red_left_eye, red_right_eye):

    image_results = hbp_nrp_cle.tf_lib.detect_red(image=camera.value)

    red_left_eye.rate = image_results.left
```

```
red_right_eye.rate = image_results.right
```

LISTING 4.1: An example of transfer function code, translating an image into spike rates.

In this example, it can be seen that through the use of the decorators DSL several properties are specified, such as the type of TF (Robot to Neuron), the devices towards the brain simulation (spike generators firing with Poisson statistics attached to the neuron population) and the input coming from the robotic simulation (camera image published through a ROS topic). It can also be noticed that the actual business logic is implemented inside the function, and in particular, the image is processed with a color detection filter implemented as part the TF library provided alongside the platform.

The choice of Python for the TF framework was the most natural one, given the fact that both the chosen physics and neural simulators provide Python APIs. Consequently, the rest of the server side NRP components have been written in Python. In principle, this could raise performance issues when compared with languages like C++. It was chosen to avoid fine tuning of the performance of the developed components, as currently the bottlenecks of a simulation reside in the physics and neural simulators. This choice has also the advantage of simplifying considerably the development process.

Internally, the complete BIBI configuration, comprising the transfer functions, the robot model and the brain model, is stored as an XML file. Each transfer function can be saved either as Python code in an XML tag or can be constructed from custom XML elements which are later parsed in order to generate the equivalent Python code. The second way of describing these functions is better suited for the automatic generation of such XML files, via graphical editors that could be used also by scientists with no experience in Python.

4.3.4 Closed Loop Engine

The *Closed Loop Engine* (CLE) is responsible for the control of the synchronization as well as for the data exchange among the simulations and the TFs. The purpose of the CLE is to guarantee that both simulations start and run for the same timestep, and also to run the TFs that will use the data collected at the end of the simulation steps. Figure 4.4 shows a sequence diagram of a typical execution of a timestep: after the physics and neural simulations have completed their execution in parallel, the TFs receive and process data from the simulations and produce an output which is the input for the execution of the next step. The idea behind the proposed synchronization mechanism is to let both simulations run for a fixed time step, receiving and processing the output of the previous steps and yielding data that will be processed in the future steps by the concurrent simulation. In other words, data generated by one simulation in the current timestep cannot be process by the other simulation until the next one. This can be read as the TFs introducing a delay of sensory perception and motion actuation greater than the simulation timestep.

The use of MUSIC for the synchronization was excluded in this first release, even if it was shown to be working by (Weidel, Duarte, et al., 2015; Weidel, Djurfeldt, et al., 2016), in order to ease the communication between brain and world simulations without introducing any middle layer. Moreover, relying on the already existing Python APIs for the communication with the two simulators, had the effect of simplify the development process.

Besides orchestrating running simulations, the CLE is also responsible of spawning new ones, by creating new dedicated instances of the World Simulator and the Brain Simulator, and a new instance of the orchestrator between the two.

Simulation Control

During its life cycle, each simulation transitions through several states, as depicted in Figure 4.5. At the beginning, a simulation is in state *created* and it will switch to state *initialized* once the CLE is instantiated. Up to this point, no simulation steps have been performed yet. Once the simulation is *started*, the CLE will start the interleaving cycle, that can be temporarily interrupted by pausing the simulation (*paused*) or preemptively terminated by stopping the simulation (*stopped*). If any error occurs during the execution or during the transitions between states, the simulation will pass automatically to the state *halted*. The *reset* transition can be considered parametrized, as it allows restoring to their initial status separate parts of the simulation singularly. Currently, the resettable parts in a simulation are the robot pose, the brain configuration and the environment.

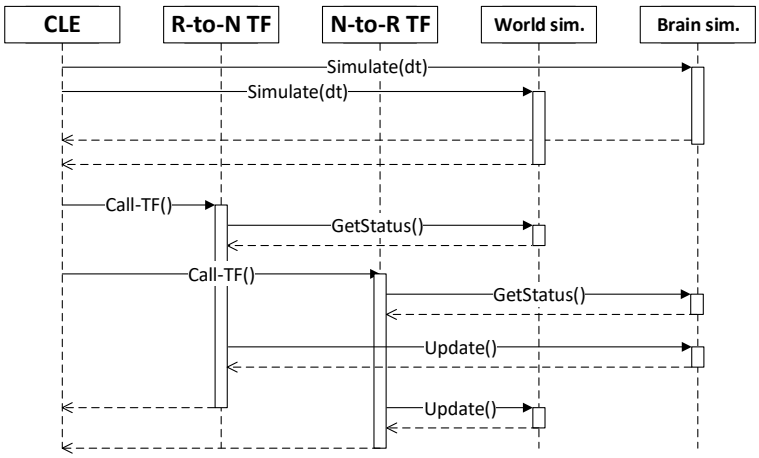


FIGURE 4.4: Synchronization between the components of a simulation, as orchestrated by the CLE. In a first phase, the two simulations are run in parallel. Afterwards, each transfer function gathers data from simulations and compute the appropriate inputs for the next simulation step.

Thanks to the possibility of pausing and restarting the closed loop cycle during the simulation execution, it was possible to add features that modify simulation properties at run-time, without the need to restart the simulation from scratch. These features include support for transfer function adding, editing and removal, brain model and environment editing. Using these features, it is possible to test different configurations of the simulation and immediately see the effects of them, without having to wait for a complete restart.

From the point of view of the implementation, the timestep of the physics simulation is sent to Gazebo through a ROS service call, while the brain simulation is directly run for the desired timestep with a PyNN call, as it can be observed from the architecture depicted in Figure 4.6. Access to ROS service calls and PyNN calls relative to the control of the simulation are achieved by implementing generic adapters interfaces, performing a client-server interaction. Hence, in principle, a CLE instance can interact with different simulators than the ones currently supported (Gazebo and

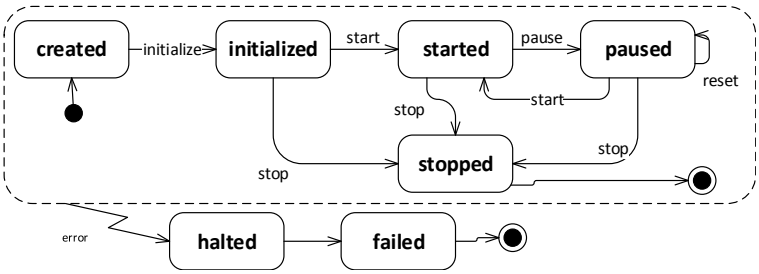


FIGURE 4.5: Lifecycle of a simulation in the NRP. During a normal cycle, the simulation will start from the *created* state, passing through *initialized* as soon as the resources are instantiated, then going through the *started* state once the execution is initiated, and finally in the *stopped* state. During the execution the simulation can be *paused* at any time, while if any error occurs during the normal lifecycle the simulation is *halted*.

NEST). This abstraction layer, besides providing the possibility to change with relative ease the underlying simulators, simplifies the update process of the simulators, by limiting the number of files that need to be changed in response to a possible API update.

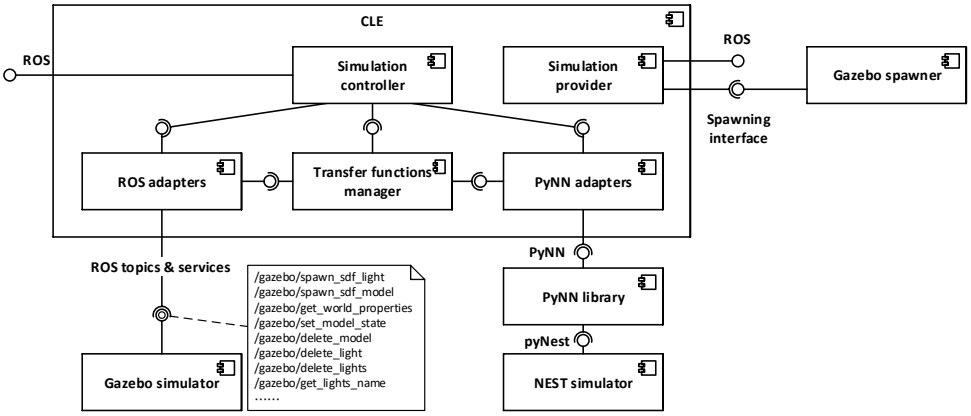


FIGURE 4.6: Architectural overview of the CLE and of the communication layers. The CLE orchestrates the two simulations and performs the data exchange through generic adapter interfaces. It also provides two interfaces, one for controlling an ongoing simulation and one for providing a new one, by instantiating a neural simulator and a physics simulator. In the current implementation, adapters for accessing Gazebo physics simulation via ROS and for accessing NEST neural simulation via PyNN are provided. In particular, robot data is accessed through ROS services and topics and the physics simulation is controlled through ROS services.

State machines for simulation control

In real experiments, it is often the case that the environment changes in response to occurring events, generated by the behavior of the subject, by the experimenter or automatically generated (i.e. timed events). Thus, in order to reproduce this behavior, the possibility to generate events that can influence the environment was

added to the platform. In particular, the user can interact with some objects without having to interrupt the simulation, like changing the brightness of lights or screen colors, and an event system is provided. The event system is implemented with a state machine that is programmable by the user. In the current implementation, support for timed events is provided, allowing the user to program changes in the environment that have to occur at specific points in time.

The event system is managed by the *State machines manager*, implemented using the SMACH state machine framework (Bohren and Cousins, 2010) that is already integrated into ROS. Using such a framework, it is possible to program timed events that directly call Gazebo services in order to modify the environment.

4.3.5 Backend

The *Backend* is the component connecting the user interface to the core components of the platform, exposing a web server implementing RESTful APIs on the user interface end point and forwarding processed user requests via ROS on the other end point. This component is the first handler for user requests. In case they could not be completely managed within the backend, they are forwarded either to the CLE or to the State machines manager, that will eventually complete the request processing, interacting, if necessary, with the simulators. An overview of the Backend architecture and of the interaction with other components is depicted in Figure 4.7.

Actions provided by the backend to the user interface (ESV) include experiment listing and manipulation, simulation listing, handling and creation, and gathering of backend diagnostic and information.

Every available experiment on the platform is identified by a name and a group of configuration files, including a preview image to be showed on the ESV and files representing environment, brain, state machines and BIBI, where neural populations and transfer functions are stored. Experiment listings and manipulation APIs allow the user to list all the available experiments on the server as well as retrieving and customize singularly the configuration files of the experiment.

In the NRP setting, a simulation is considered as an instance of one of the available experiments. In order to create a new simulation, the user has to proceed in a different way depending on whether the NRP is accessed within or outside of the Collaboratory Portal. If users are accessing from the Collaboratory Portal, they are able to clone the configuration files related to one of the available experiments on the Collaboration storage they are using, and instantiate that local copy of the experiment. The backend allows users to overwrite said configuration files as well as saving CSV recordings of simulation data directly on the storage. In case a user is not working from the Collaboratory Portal, they can instantiate an experiment choosing directly from the experiment list, and they can edit it without having to instantiate a local copy.

Once a simulation is created, the backend allows the user to retrieve and change its current state according to the simulation lifecycle depicted in Figure 4.5, by interfacing with the CLE. Other APIs provide functionalities for retrieving and editing at runtime the brain configuration, the state machines and the transfer functions, delegating again the task to the CLE. Furthermore, information about the simulation metadata, brain populations and environment configuration is available through dedicated APIs.

For diagnostic purposes, the backend provides some APIs for retrieving the errors which have occurred on the server as well as the version of the backend itself and the CLE.

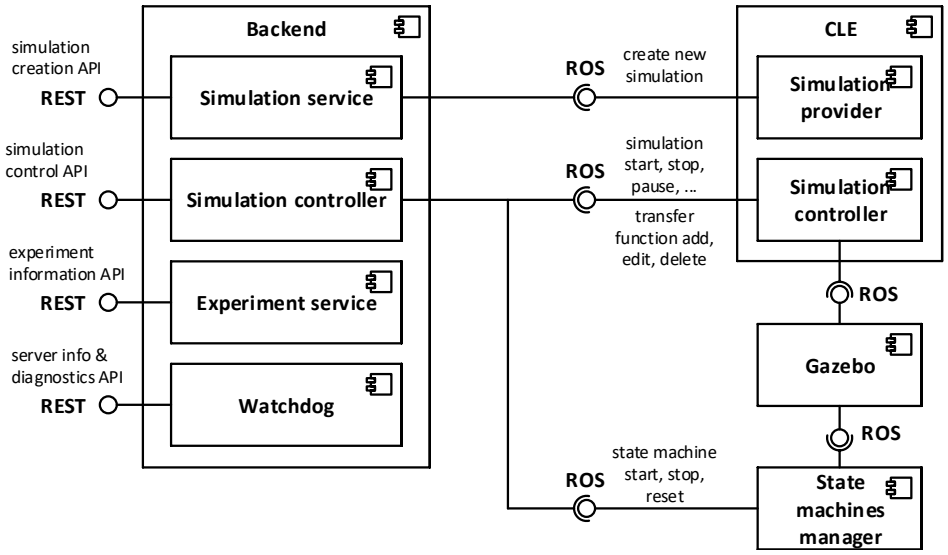


FIGURE 4.7: Architectural overview of the Backend. User inputs coming from the ESV are sent to the Backend via REST APIs. These request are then dispatched to the CLE or to the State machine manager that will handle them by forwarding the appropriate commands to Gazebo.

4.3.6 Experiment Simulation Viewer

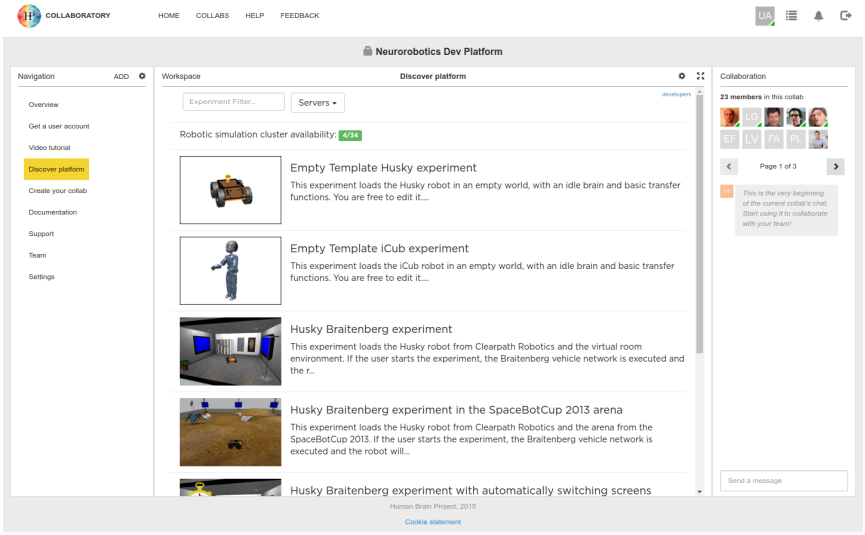
The Experiment Simulation Viewer is the user interface to the NRP. It is implemented as a web based application, developed using a modern web software stack exploiting established open source software. The ESV is currently integrated in the Collaboratory Portal (see Figure 4.8a) using the Collaboratory APIs. By building it using standard web technologies, cross-platform support, also for mobile devices, is enabled. The downside of this choice is the added complexity of using translation layers, albeit lightweight ones, for the interaction with server side components.

The ESV simulation interface embeds a 3D view that allows the user to see and navigate through the virtual environment, and a user bar for simulation control (e.g. for playing, pausing, resetting, or stopping the ongoing simulation). It also provides means for editing objects by altering their attributes, and monitoring brain activity and the state of the embodiment, on a running simulation. Furthermore, the simulation interface hosts the tools that allow the user to design and edit an experiment, explained in depth in Section 4.3.6. Any modifications to the running simulation can be exported either on the user computer or saved on Collaboratory storage.

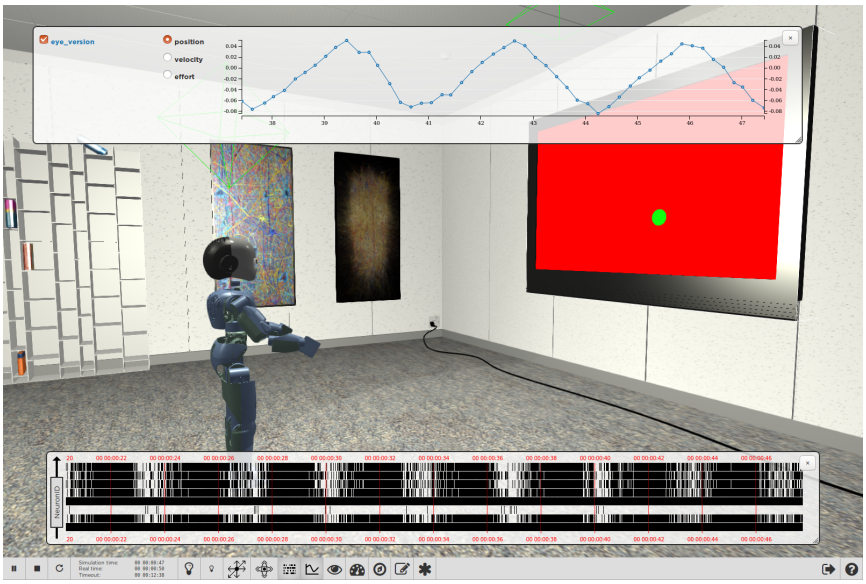
User Interface

Entering the ESV, the user is presented with a list of available experiments (see Figure 4.8a). For each experiment, the user can choose to launch a new simulation, or to join an already launched one as a spectator; it is also possible to launch an instance of an existing simulation while uploading a custom environment in which it will be executed, thus replacing the original one.

The user starting a simulation is called the “owner” of that simulation whereas any other user is called a “watcher”. The owner has full control over his simulation,



(a)



(b)

FIGURE 4.8: The Experiment Simulation Viewer. Through the Collaboratory Portal, depicted in (a), the user can choose between several predefined experiments or create his own experiment based on templates. Some of the features of the Collaboratory Portal such as a navigation pan and a group chat are shown. (b) shows the ESV Main View, where an experiment is being executed. The user interface at the bottom allows the user to interact with the simulation and displays information about simulation time. Some widgets, togglable from the user bar, allow the user to monitor brain activity in term of spike trains or joint values of the simulated robot.

being able to start, pause, stop or reset the simulation and interact with the environment while it is running. Other features like monitoring or navigation into the scene are accessible to both owners and watchers.

Of particular interest are the monitoring features (Figure 4.8b). The Spike Monitor plots a spike train representation of the monitored neurons in real-time. Monitored neurons must be specified by transfer functions, as described in Section 4.3.3.

The Joint Monitor plots a line chart representation of the joints properties of the robot in real-time. For every joint selected, properties like position, velocity and effort can be chosen.

The goal of these monitoring tools is to get live insights on how the simulation performs. Both spike data and joint data can also be saved in CSV format for further off-line analysis.

Architecture

In order to have a coherent user interface and experience throughout, all the tools developed in the Human Brain Project, including the NRP User interface are implemented as Web applications. An architectural overview is shown in Figure 4.9.

The application framework of choice is AngularJS[1]. AngularJS is a Model View Controller (MVC) Web Framework for developing single-page applications. Using AngularJS services, the interaction with the NRP Backend, which provide the API for the simulation control, is realized via standard REST calls.

The Rendering of the 3D view of the virtual environment is performed by Gzweb, Gazebo's WebGL client. It comprises two main parts: gz3d and gzbridge, which are, respectively, responsible for visualization and for communicating with Gazebo's backend server gzserver.

To enable the communications with the CLE and Gazebo via ROS, the ESV employs roslibjs, a JavaScript library. Roslibjs in turn interacts via WebSockets with rosbridge, a tool providing a JSON API to ROS functionality for non-ROS programs.

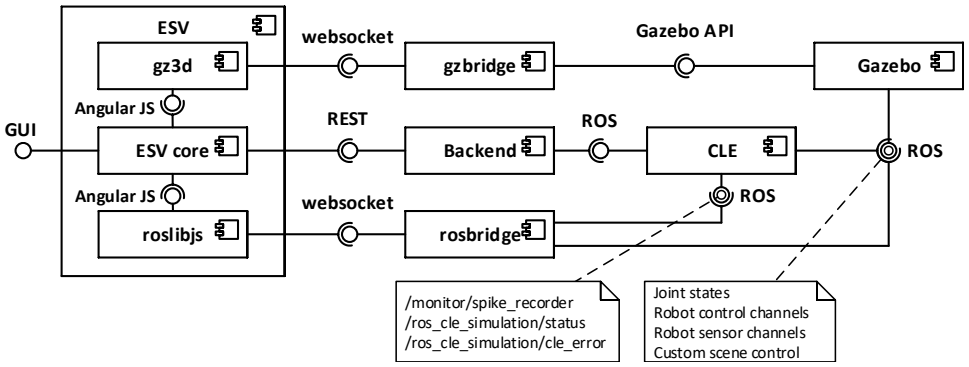


FIGURE 4.9: Architectural overview of the ESV and its outgoing communications. The ESV core interacts with the Backend via REST APIs to control the simulation. Two websocket channels bypass the Backend and allow the ESV to interact directly with Gazebo and the CLE. The gzbridge channel is used for gathering information about the scene rendering. The rosbridge channel is used for collecting information related to brain spikes and joint angles, that is shown on appropriate monitoring widgets in the GUI, and to get information about simulation status and possible errors.

Editors

In order to design the experiment to be simulated, the NRP provide the user with a complete array of tools. All these tools are seamlessly integrated into the ESV application, thus switching between simulation and editing is a very fast process:

[1] https://www.angularjs.org/

the user can immediately simulate the modified experiment, and if not satisfied, directly modify it again. Thanks to these tools, it is possible to configure all the aspects of an experiment:

- **Environment**:
  The purpose of the Environment Editor is to allow the user of the platform to set up the scene in which the simulation will run, either starting from scratch or editing one from an existing experiment. While running the environment editor, the user is able to move (e.g. translate or rotate) or delete existing objects in the scene, or to place new objects by choosing them from a list of models. The modified scene can be exported as into the Simulation Description Format (SDF)[2], for later reuse.

- **Transfer Functions**:
  The Transfer Functions (TFs) describe how to transform simulator specific sensory data (such as image, joint angles, forces, etc.) to spiking activity for neural network simulation and vice versa. TFs are defined as Python scripts exploiting the DSL described in Section 4.3.3. From within the editor, the user can create and edit TFs as well as save them to and load them from files. Every uploaded transfer function is checked for syntax errors, and several restriction for Python statements are applied for security reasons. Furthermore, the user can log TFs' data to files using the standard Comma Separated Values (CSV) format. Like for the other editors, the edited TFs can be downloaded and stored.

- **Brain**:
  The Brain Editor allows the user to upload and edit custom brain models as PyNN scripts (Davison et al., 2008). The PyNN script describing the brain model used in the current experiment, is shown in a text editor featuring Python keyword and syntax highlighting. It is also possible to define populations (i.e. sets of neurons indices) that can be referred to in transfer functions. Edited brain models can be saved.

- **Experiment Workflow**:
  The workflow is specified in Python code exploiting SMACH and the Experiment Workflow Editor allow to create such state machines, as well as saving them for later reuse.

As for the editing of the robot model, the development from scratch of a custom software (either web or desktop) for modeling and designing a robot is an enormous undertaking, so it was decided to adopt existing solutions. In particular, no reasonable web solutions were found, and adapting existing solutions for web would require a considerable effort which would not be counterbalanced by the possible benefits. It was then chosen to use Blender[3] (a powerful and extendable open source software) among the existing modeling softwares, due to its availability for a wide range of platforms with a simple installation process. The *robot designer* is thus provided as a Blender plugin that provide users with an easy-to-use user interface that allows the construction of robot models and defining kinematic, dynamic and geometric properties.

[2] http://sdformat.org/
[3] https://www.blender.org/

Repository	total lines	tests	line coverage	branch coverage
CLE	2944	147	88%	100%
Backend	3045	239	93%	100%
Frontend (ESV)	2427	455	95%	87%
ExperimentControl	455	46	96%	100%

TABLE 4.1: Summary of quality control statistics for the NRP repositories, at the time of the first public release.

4.4 Software development methodology

The NRP is developed within the Scrum agile software development framework (Schwaber and Beedle, 2002). The basic unit of development, in Scrum parlance, is called a *Sprint*. It is a timeboxed effort, that is restricted to a specific duration of either two or three weeks. This methodology provides a reactive environment, able to deal with changing requirements and architectural specifications.

The Scrum process includes daily stand-up meetings, where each team of developers discusses about how they are working to meet the sprint goals. At the end of the sprint, a review meeting is held; the whole NRP team is present, and the members make demonstrations of the software in stable development status. Each completed task provides a new feature to the user, without breaking compatibility with the current code base. Thus, at the end of each sprint, there is a new shippable platform that provides new features.

The NRP software process uses industry standards for quality control. The acceptance criteria of the version control system includes the necessity of a code review by, at least, a second programmer, while a continuous integration system ensures that new code does not introduce regressions by executing a set of unit tests. Moreover, code coverage criteria ensures that at least 80% of the code is covered during tests and coding standards are enforced by automatic static code analysis tools (PEP8 and Pylint). Each build in the continuous integration system also produces the software documentation documenting the APIs and comprising software usage examples. A summary of quality control statistics regarding the main NRP repositories, at the time of the first public release, is presented in Table 4.1. No repository has PEP8 or Pylint errors.

4.5 Use cases for the Neurorobotics Platform

In order to assess the functionalities of the NRP, some experiments were designed. These experiments, albeit simple in nature, aim at demonstrating various features of the platform. The first use case is just a proof of concept: a very simple brain model is connected to a robot via TFs in order to have a complete action-perception loop performing a visual tracking task. Results show that the two simulations are properly synchronized and the experiment is correctly performed.

The second one demonstrates the extensibility of the framework: an already existing computational model of the retina was integrated inside the platform and used to perform bio-inspired image processing. This experiments also moves towards finding generic translation mechanism that can be used as Transfer Functions.

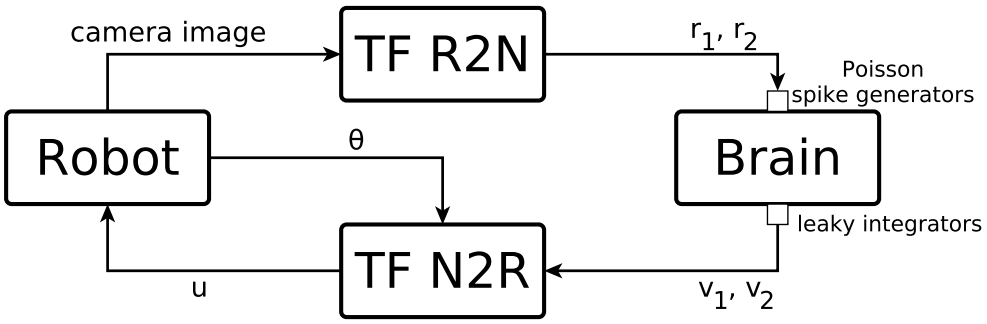


FIGURE 4.10: Tracking architecture as implemented inside the NRP.

4.5.1 Basic Proof of Concept: Visual Tracking

As an example to demonstrate the applicability of spiking neural networks for movement generation, a tracking experiment that shows how a humanoid robot can be controlled by an "artificial brain" was implemented. In particular, the focused was put on a visual tracking task. A humanoid robot stands in front of a blue screen with a moving green circle which is the target of the tracking. The robot reaches and follows such a target using only eye movements. In this task it uses only horizontal eye movements, but the same controller can be easily extended to the vertical axis. The humanoid robot used for this task was the iCub robot.

Materials and methods

The available sensors in the model are two 320X240@60Hz cameras for the eyes and encoders for each joint. In order to control each joint a gazebo plugin was written using the Gazebo API. Each joint is controlled by two PID, one for positions and one for velocities, both implemented using the Gazebo API library functions. The plugin publish the encoders values to a ROS topic and subscribe to other ROS topics to receive position and velocity commands. The cameras were implemented using the standard Gazebo camera plugin.

During the experiments the eye version was controlled in position. This virtual joint is directly connected with the eye pan real joints. The gains used for the velocity PID controller of each eye pan joint were the following:

$$P = 2.0, \quad I = 0.1, \quad D = 0.003$$

To implement a controller for the aforementioned task, information about the target position is extracted from the camera image and then sent to a controller which generates appropriate motor commands to move the gaze towards the target.

To implement such a model in the Neurorobotics Platform, the target extraction must be implemented as Robot to Neuron transfer function, while the controller must be implemented as a brain model plus an appropriate Neuron to Robot transfer function (Figure 4.10).

In the *Robot to Neuron* transfer function the sensory data coming from the robot is translated into an input for the brain model: in this case the input is the camera image. From this image, the target position is extracted via a simple colour filtering. Then, only information about the fact that the object is in the left or right side of the image is sent to the brain. In order to do this, the angle of the object in the camera

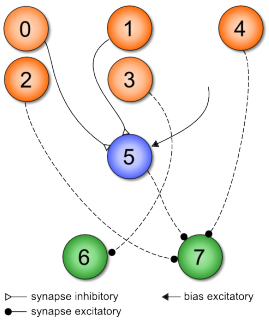


FIGURE 4.11: Brain network. Neurons 0 to 4 are input neurons, while 6 and 7 are output neurons.

reference frame is flattened into this kind of information with a sigmoidal logistic function:

$$r = \frac{1}{1 + e^{\alpha}} \tag{4.1}$$

where α is the angle of the object in the camera reference frame and r is closer to 0 the more the object is on the left and closer to 1 the more the object is on the right.

This information is sent to the brain by using two Poisson spike generators which rates are computed as follows:

$$r_1 = 1000 \cdot r \tag{4.2}$$
$$r_2 = 1000 \cdot (1 - r) \tag{4.3}$$

The *brain model* was implemented as a network of 8 adaptive exponential integrate and fire neurons (Brette and Gerstner, 2005), connected as depicted in Figure 4.11.

Neurons 0 to 3 receive input from the first spike generator coming from the robot to neuron transfer function, while only neuron 4 receives input from the second spike generator.

The neurons labelled as 6 and 7 are the output neurons that will send the output to the neuron to robot transfer function. Given the network configuration, these neurons codify the direction towards which the eye should move. The synaptic weights of the different connections were hand-tuned to produce the best experiment results.

In the *Neuron to Robot* transfer function, the output neurons are connected to two leaky integrators, implemented with inhibited integrate and fire neurons. Thus, the input of this transfer function are the two membrane potential of said neurons, v_1 and v_2.

The motor command given to the robot is then computed as a sum between the current eye position and a displacement given by a function of v_1 and v_2:

$$u = \theta + f(v_1, v_2) \tag{4.4}$$

where θ is the eye joint angle and f is defined as

$$f(v_1, v_2) = k - 2 \cdot \frac{v_2 - v_1 + 0.03}{0.09} \cdot k \tag{4.5}$$

where k is a parameter defining the absolute maximum displacement.

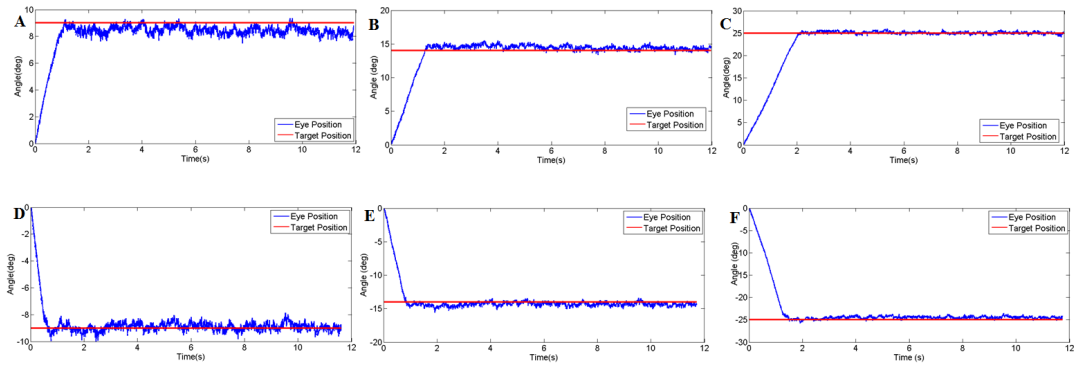


FIGURE 4.12: Controller performance in the step response (target on the left (top) and target on the right (bottom). The considered target positions are: 9 degrees (left), 14 degrees (middle), 25 degrees (right).

During simulations, a value of $k = 0.5$ was used. The value of u is then sent to the robot as a position motor command for eye vergence.

Results

Two experiments were performed: a step response movement and the actual visual tracking task.

In the *step response* task, the controller had to reach a static target from a starting position (eye version equal to 0). Figure 4.12 shows the eye motion needed to reach a static target positioned on the left side (Figure 4.12, top) and on the right side (Figure 4.12, bottom) of the camera image, at a rotation distance of 9,14 and 25 degrees (eye rotation needed in order to centre the target in the camera image). It can be observed that the time needed to reach the target is dependent on the motion amplitude (less than 1 second for 9 degrees and almost 2 seconds for 25 degrees motion). In addition it should be noticed that the performances of the proposed controller do not depend on the motion side (the time needed to reach the target on the left and on the right is not statistically different). The mean square error (comparing eye position and target position) in the last second of the trial ranges from 0.06 for the left motion with an amplitude of 14 degrees to 0.14 of the left motion with an amplitude of 9 degrees. There is no relationship between the mean squared error and the motion direction or the motion amplitude.

In the *visual tracking*, the robot had to follow a target displayed on a screen moving with a sinusoidal motion (see Figure 4.13). The performances of the model were tested at various frequencies (from 0.1 to 0.3 Hz), with an amplitude of 18 degrees. It can be observed that the model is able to pursue the target. Figures 4.14 and 4.15 show the pursuit of the target, for oscillation frequencies of 0.2Hz and 0.3Hz, by comparing the target and current eye horizontal positions in time. In addition, the robot is able to follow the target at 0.2Hz with just a delay but without any reduction in amplitude (the eye motion amplitude compared to the target one), while at 0.3Hz the performance deteriorates. The mean square error increases from 0.36 at 0.1Hz to 0.46 at 0.2Hz and 0.72 at 0.3Hz.

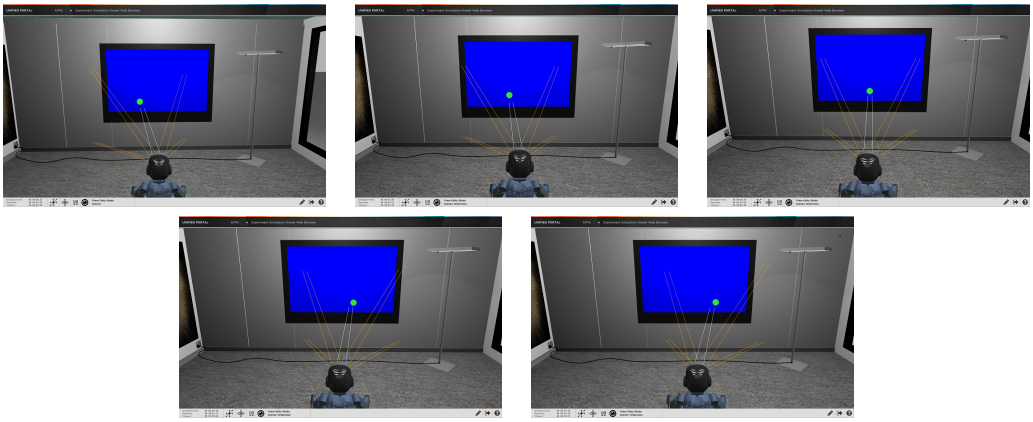


FIGURE 4.13: Snapshots of the iCub performing visual pursuit inside the Neurorobotics Platform.

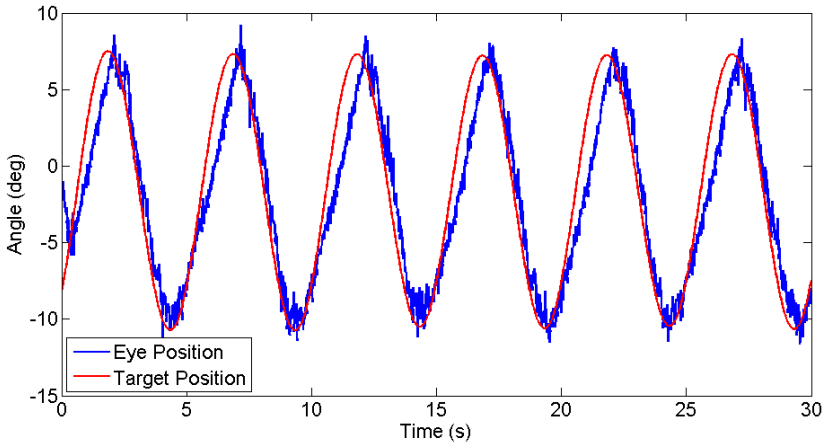


FIGURE 4.14: Target and eye alongside the x axis during an oscillation frequency of 0.2Hz.

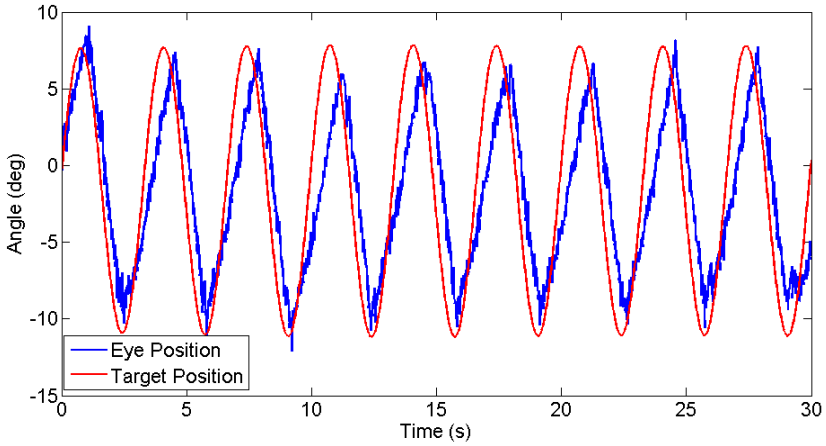


FIGURE 4.15: Target and eye alongside the x axis during an oscillation frequency of 0.3Hz.

4.5.2 Integration of Bio-inspired Models: Retinal Vision

In order to have full biologically inspired closed loop controllers, the transfer functions should also make use of neuroscientific models of sensor information processing from one side and motion generation on the other. This also provide more reusable translation approaches for sensory information, compared to tailor-suited ones, like the one presented in the previous example. As a first step in this direction, a retina simulator was included in the NRP as a robot to neuron transfer function. This allowed to create a more complex controller for the eye that employ the red-green opponency mechanism (Dacey and Packer, 2003).

Integrating the retina simulation platform in the NRP

The model chosen for the integration was *COREM*, a computational framework for realistic retina modelling (Martínez-Cañada et al., 2015; Martínez-Cañada et al., 2016), that provides a general framework capable of simulating customizable retinal models. In particular, the simulator provides a set of computational retinal microcircuits that can be used as basic building blocks for the modelling of different retina functions: one spatial processing module (a space-variant Gaussian filter), two temporal modules (a low-pass temporal filter and a single-compartment model), a configurable time-independent non-linearity and a Short-Term Plasticity (STP) function.

The original retina codebase (Martínez-Cañada et al., 2015) was implemented as a stand-alone program. So the first step towards the embedding in the NRP has been to isolate all the core functionalities of the model in a separate module, that could be used as a library. The original executable has been kept as a "frontend" application depending on such library.

In order to have a retina model involved in the robot control inside a NRP simulation, it is necessary to forward the camera image or the robot to the retina, get the retina output information so it can be processed in a SNN, and finally decode the SNN output to control the robot. All these steps must take place inside transfer functions.

The whole retina model is written in C++, whereas all the NRP backend, including transfer function related business logic, is written in Python. Thus, to allow the NRP to send input to the retina and get output from it, the functions in the retina library involved in these processes have been wrapped with Python bindings, making them accessible from Python code.

Retina based controller

The retina-NRP interaction was tested by creating a custom experiment on the platform performing a visual tracking task with a humanoid robot. Since the controller employs the red-green opponency mechanism, an environment in which a simulated iCub humanoid robot will have to track a green ball moving horizontally on a red background was created.

In the experiment, images from one of the two cameras mounted on the robot eyes are sent to the retina simulator. Retinal output is then gathered and one horizontal stripe of the output image (320 pixels from a 320x240 image), intersecting the target, is forwarded to a neural network, where the spikes are filtered in order to avoid any undesired false detection. Finally, spike trains coming from the SNN are translated in a motion command for the robot eyes, making them following the target. An overview of the controller is shown in Figure 4.16. Using a single camera of the robot reduces the computational times as only one retina model can be

employed but it does not permit to control the vergence of the two eyes. Therefore, albeit both eyes perform the same motion, the tracking is actually only monocular.

The *retina circuit*, implemented with the framework described above, is composed of two symmetric subcircuits processing independently the input image. In the first layer we can find simulated L-cones and M-cones, receiving the initial image. Output from the photoreceptors is then forwarded to the two different subcircuits, namely the L^+M^- circuit and the L^-M^+ circuit.

The L^+M^- circuit (L^-M^+ description can be derived by symmetry) sums the output signals from the two cones, giving a positive weight to L-cones and a negative weight to M-cones. Combined cones signal is processed by simulated horizontal, bipolar, amacrine and ganglion cells. Every simulated ganglion cell is then sensible to a red center on green background within its receptive field. The high level result is a sensitivity to borders on a static image, which is slightly accentuated around borders between red and green objects for both circuits. In case of moving images instead, ganglion cells in the L^+M^- are particularly sensitive, due to their temporal characteristics, to green objects appearing in receptive fields that were earlier impressed by red objects. Thus, the output is information about the changing of colour in the image. This peculiarity will be exploited to infer the position of an object by combining responses from the two circuits.

The *brain model* used in the experiment comprises 1280 integrate and fire neurons (Brette and Gerstner, 2005), organized in two layers. The first layer acts as a current to spike converter, it has been designed for taking current intensities value coming from the retina ganglion layer to spikes, while the second one filters the spiking ganglion output to remove noise such as the one produced by the motion of the eyes. In the second layer, every neuron, except the "side" ones, gather information from 7 subsequent neurons on the first layer, acting as a local spike integrator.

Every layer embeds two independent populations of 320 neurons, processing separately output from the two different ganglion cell layers of the retina circuit, so there is a one to one correspondence between considered pixels and neurons in a single first layer population. Thus, neurons from 1 to 320 will get input from the first ganglion circuit and forward spikes to neurons from 641 to 960, and neurons from 321 to 640 will get input from the second ganglion circuit and forward spikes to neurons from 961 to 1280.

The summation occurring in the second layer, together with linearly decreasing synapse weights from the center of the image to the periphery, serves as a filter for avoiding undesired spikes in peripheral regions of sight.

The robot controller is implemented by means of two *transfer functions*, one robot to neuron and one neuron to robot.

In the robot to neuron TF, the image on the eye camera of the robot is collected and forwarded to the retina library, updating its status. Outcome from the ganglion layers of the retina circuit are processed, then current values for a strip of pixel containing the target are transmitted to the two populations in the first layer of the brain.

Output spikes from the second brain layer are then processed by the second TF with the following steps:

[3]Neuron image from Amelia Wattenberger, released under Creative Commons License 3.0 https://thenounproject.com/term/neuron/79860/

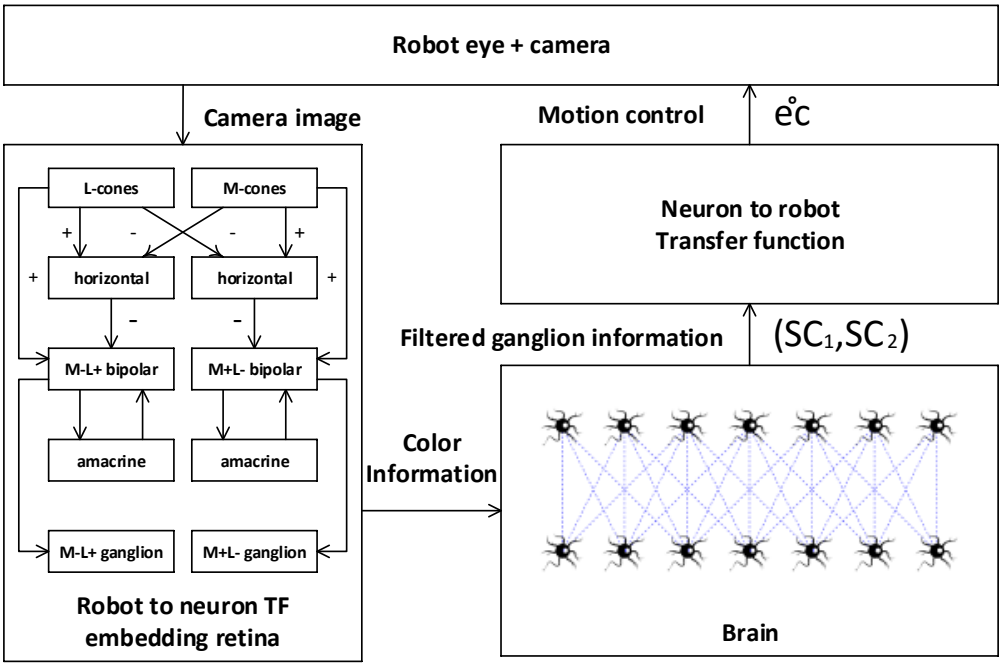


FIGURE 4.16: Block diagram of the implemented robot controller. The retina model, implementing the red-green opponency circuitry, process the images and outputs information about the perceived changes of colour in the pixels (green over red and vice versa). The brain model converts the analogue output of ganglion cells into spikes and filters it to remove noise.

- Spike counts for every neuron on the two output population are gathered in two collections SC_1 and SC_2, that are represented as two functions

$$SC_1, SC_2 : [1, 320] \rightarrow \mathbb{N}$$

where $[1, 320]$ is a discrete set, representing a neural population.

- As higher stimulation of ganglions correspond to higher spike frequency in the brain, one clearly distinguished frequency maximum per population is expected, the first one corresponding to the pixel where the ball enters the receptive field in the first ganglion layer and the other to the pixel where the ball leaves the receptive field in the second ganglion layer. Since one neuron per pixel on an horizontal stripe of pixels is being used, the two maxima will correspond to two column indexes of the original image, which can be called p_1 and p_2. For each population, the said column index is obtained by computing first the maximum neural spike rate, then taking the first neuron index with that spike rate.

  In formulas, p_1 and p_2 are defined as

$$p_1 = \min\{\arg\max_x\{SC_1(x)\}\}$$

$$p_2 = \min\{\arg\max_x\{SC_2(x)\}\}$$

- The estimate of the target center the value can be taken as

$$\bar{p} = \frac{p_1 + p_2}{2}$$

- The estimated position of the ball $\bar{p}$ is then converted to an estimated angle $\mathring{p}$ with respect to the eye center with the equation

$$\mathring{p} = -\arctan\left(\frac{\bar{p} - 160}{160}\right)$$

- The eye position angle command $\mathring{ec}_{t+1}$, depending on the current eye position $\mathring{e}_t$ is computed as follows

$$\mathring{ec}_{t+1} = \mathring{e}_t + 0.2\mathring{p}$$

  where the constant 0.2 is determined empirically and prevents quick eye movements that could result in the target loss from the robot sight.

Results

As a first assessment, the controller has been validated by testing only its target detection capabilities without any actuation on the robot. The experiment was started with a slightly different neuron to robot transfer function which didn't send any control message back to the robot and with the target ball moving with a sinusoidal trajectory with a cycle frequency of $0.5Hz$, covering almost half the field of vision of the camera. During the experiment, data about both target perceived location and the neural network response was collected and it is reported in Figure 4.17.

At the beginning of the simulation, the robot suddenly moves to a default position, while the controller responsible for the upright position starts. For this reason, the camera image during the first 2 seconds of simulation may be affected by this movement and thus its elaboration may provide wrong results. In Figure 4.17a we can observe how the estimated target position, after the robot stabilization, follows a sinusoidal trend. The target position is plotted with a dashed line as a reference. Sensitivity of different ganglion inputs can be noticed in the first half of the spike plot of Figure 4.17b, where the first population spikes in correspondence to the target entering the receptive field and the second spikes when the target leaves the receptive field. The filtering action of the second neural layer can be observed in the second half of the spike plot.

Finally, experiments were performed where the controller was producing motion commands, with the same setup of the previous experiment. The controller was tested with a target moving with a sinusoidal trajectory (Figure 4.18a) and with a random linear trajectory (Figure 4.18b), namely a linear trajectory changing direction (left or right) randomly every second. For the same reasons explained in the previous test, the controller needs to wait for the robot stabilization before starting actuating the eye. It can be observed how the target estimation is still effective even with a moving eye, which validates the choice of a retina simulator as a data source for the controller.

Hardware implementation

An implementation of the same experiment was also carried out on the physical iCub platform using neuromorphic hardware (SpiNNaker) to simulate the neural

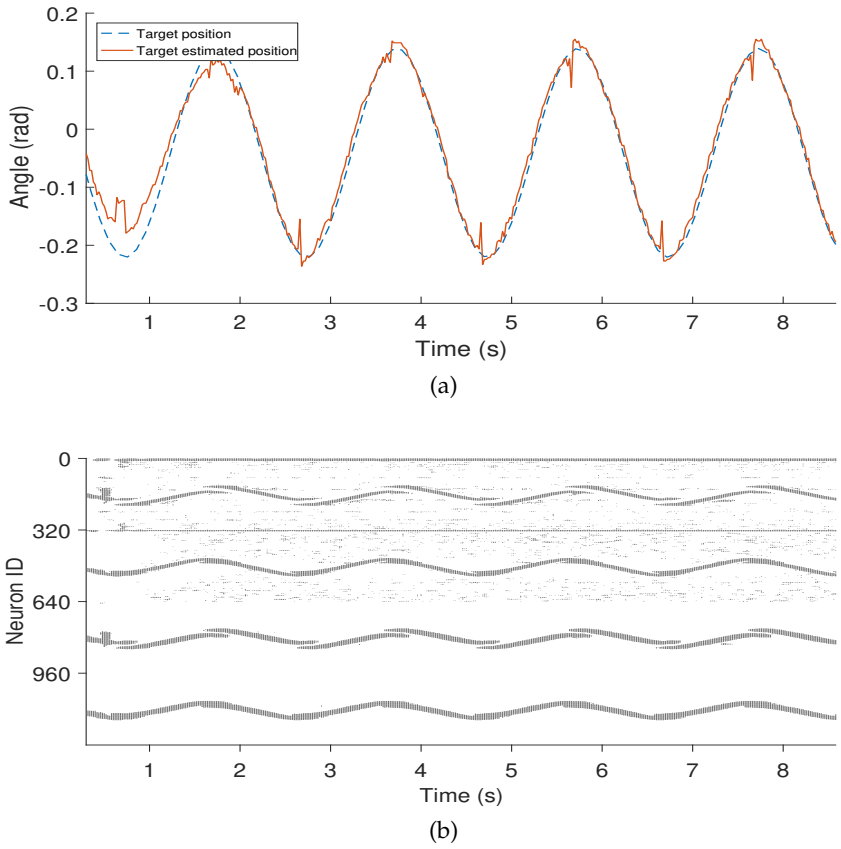


FIGURE 4.17: The computed ball position (a) and the brain response (b) for a ball with sinusoidal motion moving at 0.5Hz, robot eyes still.

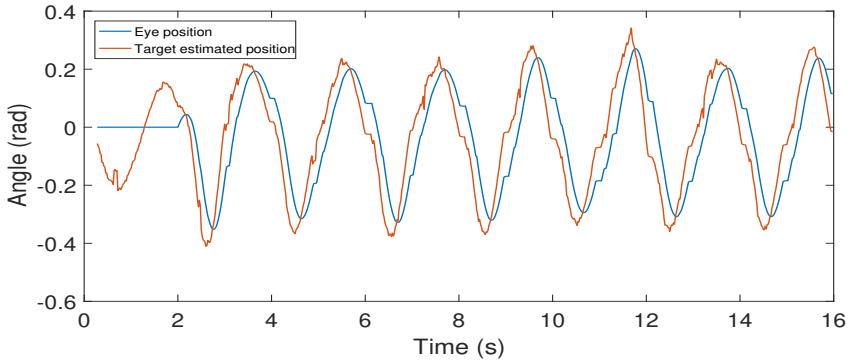
network. This experiment served as reality check to see whether such a controller could actually be implemented on real robotic platforms. To perform the experiment a similar setup was created by placing the robot in front of a red screen where an image of the green target was displayed and moved. Due to the high processing power demand of the COREM simulator, it was impossible to process images at 30Hz, as given by the camera, thus the frame rate of the retinal processing was slowed down to 20Hz, so that the real time constraints could be met. To perform the synchronization between the neuromorphic hardware and the physical robot a lightweight, real time, version of the CLE was implemented.

A comparison of the same trial performed in simulation and on the physical robot can be observed in Figure 4.19. We can observe that a similar behaviour is occurring, and that, even if the target estimation is much noisier in the hardware case, the tracking is still achieved. This proves that a translational phase, after the more rapid prototyping done in the NRP, is achievable.

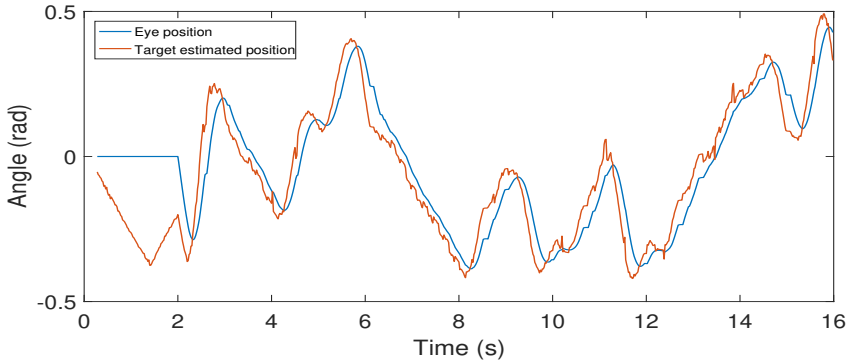
4.6 Further developments

The features detailed in the previous sections describe the first release of the Neurorobotic Platform. The development of the platform is still ongoing, in order to provide even more simulation capabilities and features to the end user.

As a personal contribution, the possibility of connecting physical robots with the NRP was implemented. This required creating an optional running mode that runs

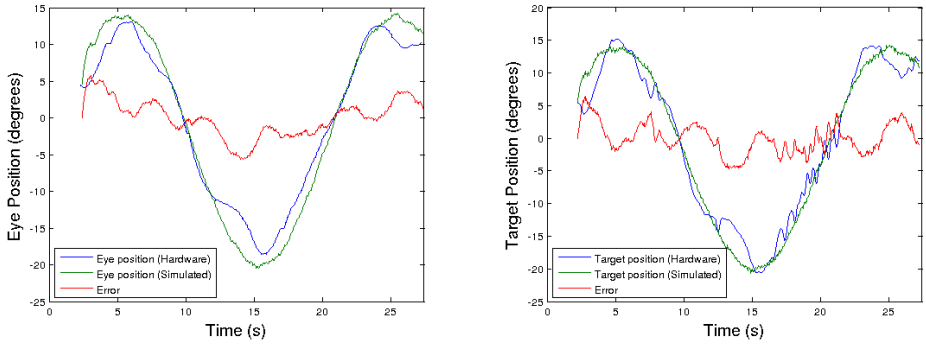


(a)



(b)

FIGURE 4.18: Tracking results with sinusoidal (a) and random linear (b) trajectories.



(a) (b)

FIGURE 4.19: Comparison between the tracking performed on the simulator and on the physical robot, eye position (a) target estimation (b).

the CLE in real time, if the computational burdens of the neural simulation and of the TF are not excessive, that avoid running the physical simulation step, and that can be interfaced with other robotic middleware such as YARP.

Another relevant development that is still ongoing is the integration of the SpiNNaker neuromorphic hardware. Currently, connection to the board is implemented with a start and stop modality, much like for NEST, thus realtime capabilities are hindered. However, a so-called "free running" mode is under discussion.

Other development plans include integration with the Brain Simulation Platform and the Neuromorphic Computing Platform, as described in Section 4.2.3, as well as an extension of the CLE that will be able to orchestrate distributed brain simulations, giving it the potential to simulate larger brain models in shorter times, that will lead to the integration with the High Performance and Data Analytics Platform.

While the retina model inclusion in the NRP provides a generic translation mechanism for images, it still has some deficiencies with respect to the biological plausibility. In particular, the processing is not space-variant, thus no foveal vision mechanism is present. To mimic this, log-polar images could be employed effectively (Capurro, Panerai, and Sandini, 1997; Traver and Bernardino, 2010).

4.7 Conclusions

This chapter presented the first release of the HBP Neurorobotics Platform, developed within the EU Flagship Human Brain Project. The NRP provides scientists for the first time with an integrated toolchain for *in silico* experimentation in neurorobotics, that is, to simulate robots with neuro-controllers including spiking neural networks in complex environments. In particular, the NRP allows researchers to design simulated robot bodies, connect these bodies to brain models, embed the bodies in rich simulated environments, and calibrate the brain models to match the specific characteristics of the robot's sensors and actuators. The resulting set-ups can permit to replicate classical animal and human experiments in silico, and ultimately to perform experiments that would not be possible in a laboratory environment. The web based user interface allows to avoid software installation and the integration within the HBP collaboratory portal gives access to storage and computing resources of the HBP. Users can run experiments alone or in team and this can foster collaborative research allowing the sharing of models and experiments.

In order to demonstrate the functionalities of the platform two experiments were performed: a simple visual tracking task and a more complex visual tracking embedding a retina model implemented on the iCub humanoid robot. These use cases make it possible to assess the applicability of the NRP in robotic tasks as well as in neuroscientific experiments. Furthermore, these experiments gave insights into the development of TFs or, more in general, of translation mechanisms to convert information exchanged between the physical and neural systems.

The final goal of the NRP is to couple robots to detailed models of the brain, which will be developed in the HBP framework. It will be possible for robotics and neuroscience researchers to test state of the art brain models in their research. At the current stage, the results achieved with the NRP demonstrate that it is possible to connect simulations of simple spiking neural networks with simulated robots. Future work will focus on the integration of the mentioned neural models. In addition to this, the integration of high-performance computing clusters and neuromorphic hardware will also be pursued in order to improve execution time of spiking neural networks replicating detailed brain models.

Chapter 5

Towards biomimetic neuro-controllers: a comprehensive spinal cord model

This chapter presents a preliminary work towards a complete biomimetic neuro-controller that should embed detailed simulations of brain areas and that could be implemented by relying on tools such as the Neurorobotics Platform described in Chapter 4. In particular this work focus on the development a generic mechanism for the translationof sensory information into neural activity and, vice versa, of neural activity into actuation command. The conversion is implemented through a model mimicking the circuitry of the mammalian spinal cord.

5.1 Introduction

In recent years, the development of action-perception closed loops that include biologically inspired neural network has risen at a rapid pace (Knoll and Gewaltig, 2016). These closed loops can be useful in both robotics and neuroscience. From a robotic perspective, these neural networks contribute to the creation of a new class of robotic controllers that could be capable of dealing with the increasing complexity of physical systems, some of which are built with hardware compliance (Tsagarakis et al., 2016) or muscle-like actuation mechanisms (Nakanishi et al., 2012). On the neuroscientific side, such loops could provide the necessary input/output connections, in a rich environment, for detailed, full-scale neural simulations that model a specific part of the nervous system, such as the cerebral cortex (Potjans and Diesmann, 2014). However, to close these loops, it is crucial to find ways to connect physical or simulated embodiments (i.e., robots or musculoskeletal systems) to these networks that mimic neural behaviors. In particular, one must find ways to translate sensory information into neural activity (i.e., spikes or synaptic currents) to connect sensors to brain model simulations, and, vice versa, neural activity into actuator commands, providing spike-based or neuromorphic translations.

In most animals, motor commands from the brain cortex are not directly sent to the muscles, but they are transmitted through a series of hierarchically organized neural circuits (Kandel et al., 2000). At the lowest level of this hierarchy lies the spinal cord. In it, information gathered from different sensory pathways (proprioceptive and exteroceptive, such as muscle spindles, Golgi tendon organs and mechanoreceptors) is collected and sent to the cortical areas, and motor commands

Part of this chapter has been adapted from the article:
Vannucci, L., E. Falotico, and C. Laschi (2017). "Proprioceptive Feedback through a Neuromorphic Muscle Spindle Model". In: *Frontiers in Neuroscience* 11, p. 341.

modulated by the brain stem are directly sent to α-motoneurons connected to the muscle fibers. Thus, we can consider the spinal cord as the place where the translation between the physical sensors and actuators is performed in vertebrate animals. Moreover, it contains neural circuits that activate and mediate a variety of basic somatic reflexes that contribute to the regulation of movements. Therefore, as the spinal cord is a such fundamental mechanism, it is only natural to consider to model it in order to develop a fundamental building block that should be embedded in complete action-perception closed loops with biomimetic neural controllers.

Several biologically realistic models of the spinal cord have been developed and tested in closed loop simulations with musculoskeletal embodiments. Stienen and colleagues (Stienen et al., 2007) developed a fairly complete model that includes Ia, Ib and II sensory afferents, both monosynaptic and polysinaptic reflexes as well as Renshaw cells, improving a previous work by Bashor (Bashor, 1998). The model was tested with a musculoskeletal model consisting of a generic antagonistic couple of muscles, thus lacking a realistic validation scenario. Cisi and Kohn developed a web-based framework for the simulation of generic spinal cord circuits with associated muscles, that aims at replicating realistic experimental conditions (i.e. electrical stimulation) (Cisi and Kohn, 2008). Sreenivasa and colleagues developed a specific neuro-musculoskeletal system, upper limb with biceps and triceps, and validated it against human recordings (Sreenivasa, Ayusawa, and Nakamura, 2016). In (Moraud et al., 2016), a simple spinal cord model of the rat, lacking any descending stimuli, was developed in order to study how such circuitry can correct the gait after a spinal cord injury and embedded in a closed loop simulation with biomechanical hindlimbs. All of the mentioned works model different aspects of the spinal circuit, however, none of them considers the modulation of spinal reflexes by γ-motoneurons, that adjust the muscle spindle's sensitivities over the wide range of lengths and velocities that occur during different tasks, a fundamental mechanism for proprioception (Banks, 1994). Moreover, these model were tested primary for the generation of reflex motions, and not as intermediate levels of more complex controller such as ones capable of generating voluntary movements.

The aim of this work is to improve on the existing spinal cord models in order to create a generic, comprehensive network, that can be used for biomimetic neural controllers and prove the feasibility of the overall approach by testing it on different embodiments, both robotics and musculoskeletric, with simulated descending inputs from the brain areas.

5.1.1 Proprioceptive feedback and muscle spindles

The brain receives and integrates information from different sensory pathways: proprioceptive, exteroceptive, and introceptive ones. In particular, proprioceptive feedback is crucial when performing voluntary movements, and its malfunction can possibly produce severe impairments such as dystonia (Kaji et al., 1995). Therefore, in the context of developing closed loops, it is necessary to properly translate proprioceptive sensory information gathered from the embodiments into a neural activity suitable for the brain model employed.

For physical robotic applications, it is suitable to embed hardware neuromorphic sensors that can natively transmit information as spike trains. Such solutions have been proved effective to translate visual information, for example by employing dynamic vision sensors to learn how to perform obstacle avoidance (Stewart et al., 2016) or to give input to vision systems integrating different eye movements (Mulas, Zhan, and Conradt, 2015; Vasco et al., 2016) and to process auditory stimuli

(Gomez-Rodriguez et al., 2007; Chan, Jin, and Schaik, 2012). However, no hardware mechanism for event-based proprioceptive feedback has been developed.

Several software solutions for generating spiking activity from proprioceptive information in closed loops have been developed. They can fall into one of two categories depending on how the translation is achieved: custom or biologically inspired ones. The first approach usually relies on tailor-made translations, specific for the tasks to be solved. In (Bouganis and Shanahan, 2010), proprioceptive feedback from the encoders of iCub arm joints is translated in the firing rate of a population of neurons by dividing the range of the joints into bins and assigning a firing rate normally distributed around the neuron encoding the current joint angle. Casellato and colleagues converted visual and sensory information on a state and an error signal that were then translated into a firing activity through radial basis functions for the control of a robotic arm, using a spiking network embedding a cerebellar model that included a learning mechanism (Casellato et al., 2014). A similar model was also used to control a single joint of a musculoskeletal robot (Richter et al., 2016). Folgheraiter and Gini presented a model of low-level reflexes for a tendon-driven hand in which analogue values for the sensors are translated into firing rates in a proportional fashion (Folgheraiter and Gini, 2004).
In contrast, biologically inspired translation approaches are developed by implementing simulations of sensory processing mechanisms found in animals, and are therefore more suitable for sending spiking activity to more realistic models of the nervous system. However, very few examples of this type exist in the current state of the art. Among these, a simple model of the muscle spindle activity has been employed to perform this translation in a closed loop between a musculoskeletal simulation and a neural model replicating low-level reflexes (Sreenivasa, Ayusawa, and Nakamura, 2016). A more complex model of the muscle spindle, albeit not completely spike-based, has been used to close the loop with a cadaver finger to create a neuromechanical system (Niu et al., 2017) and with a musculoskeletal simulation to study control of the human posture (Elias, Watanabe, and Kohn, 2014).

To provide a fully spike-based, biologically inspired translation model, it ws decided to rely on insights from biology and neuroscience. In mammals, proprioceptive information is transmitted to the central nervous system from the Golgi tendon and muscle spindle organs. In particular, the muscle spindles are the main source of proprioceptive feedback for spinal sensorimotor regulation and servocontrol. This specialized type of fiber is found inside muscles, lying along extrafusal fibers, and provide information about the length and velocity of the muscle. Several models of afferent activity coming from muscle spindles have been developed over the years. Many of these model the firing rate of the afferent fibers as a polynomial function of the muscle stretch and stretch speed (Matthews and Stein, 1969; W. Chen and Poppele, 1978; Houk, Rymer, and Crago, 1981; Hasan, 1983; Prochazka and Gorassini, 1998). These works can be classified in three groups: models based on linear transfer functions (Matthews and Stein, 1969; W. Chen and Poppele, 1978), models based on curve fitting relying on non-linear transfer functions (Houk, Rymer, and Crago, 1981; Hasan, 1983), and non-linear models relying on biological evidence of the muscle spindle (Otten, Scheepstra, and Hulliger, 1995). An interesting comparison of the firing prediction of these models, evaluated according to the hamstring spindle primary afferent firing recorded during normal stepping in cats, has been proposed by (Prochazka and Gorassini, 1998). The authors include in this work also a new hybrid model able to fit neurophysiological data more closely. All the mentioned works, albeit efficient and easy to implement, are incomplete, as they

lack two important features: response to fusimotor stimulation and distinction between primary and secondary afferent activity. In particular, fusimotor stimulation from γ-motoneurons changes the spindle's relative sensitivities over the wide range of lengths and velocities that occur during different tasks (Banks, 1994). Neuroscientific evidence indicates that this could be used in detecting changes in the desired trajectory of the intended movements, such as in locomotion (Ellaway, Taylor, and Durbaba, 2015). For these reasons, recently, more complex spindle models that include fusimotor modulation of the afferent responses have been developed (Lin and Crago, 2002; Maltenfort and R. Burke, 2003; Mileusnic et al., 2006). In the work proposed in (Maltenfort and R. Burke, 2003), the authors developed a computational model of the primary afferent activity considering the response to combinations of stretching during mixed dynamic and static fusimotor effects without considering secondary afferent activity. In (Lin and Crago, 2002), the model is more comprehensive, considering primary and secondary activities, but because of the high number of free parameters that must be tuned, it is not suitable to be integrated on different embodiments.

In this work a completely spike-based, biologically inspired mechanism for the translation of proprioceptive feedback is proposed. This mechanism implements a computational model of muscle spindle activity. In particular, the proposed model is based on (Mileusnic et al., 2006), as it is shown to be complete, including fusimotor activation and primary and secondary afferent activities, and is suitable for an implementation in closed loops (Niu et al., 2017). In principle, such a model should be used in conjunction with a detailed spiking simulation of neural pathways descending from the central nervous system that include γ-motoneuron activations. However, the original model is completely rate based. Therefore, it was extended to cope with such a spiking input, and a spike trains generation mechanism for the output of the model was added, developing a fully spike-based system. To provide a solid building block that could be used in both simulated environments and real-time scenarios, it was decided to integrate it in a spiking simulation and to implement it for a commonly used spiking neural simulator, NEST (Gewaltig and Diesmann, 2007), and for a neuromorphic hardware platform, SpiNNaker (Khan et al., 2008). To prove the correctness of the two implementations, a validation and a comparison have been performed. Finally, to prove the effectiveness of the proposed model in translating sensory information, experiments in which it has been coupled with simulated or physical systems are presented.

5.2 Muscle spindle model

The muscle spindle organ consists of three types of intrafusal fibers: bag_1 and bag_2, which are longer and nuclear, and the shorter chain type (Boyd, 1981). These types of fibers react differently to the two different types of fusimotor activations, and their activity is combined to produce primary and secondary afferent activity (Figure 5.1). Activity of dynamic γ-motoneurons only affects bag_1 fibers, while the sensitivity of bag_2 and chain fibers is regulated by static γ-motoneurons. The firing rate of primary afferent (Ia) is a combination of all the fiber activity, while only bag_2 and chain fibers contribute to the secondary afferent (II) rate. Because of the anatomy, Ia afferent endings carry information to the central nervous system that depends on both the length and stretch speed of the muscle, while II afferent endings provide information relative mostly to the length.

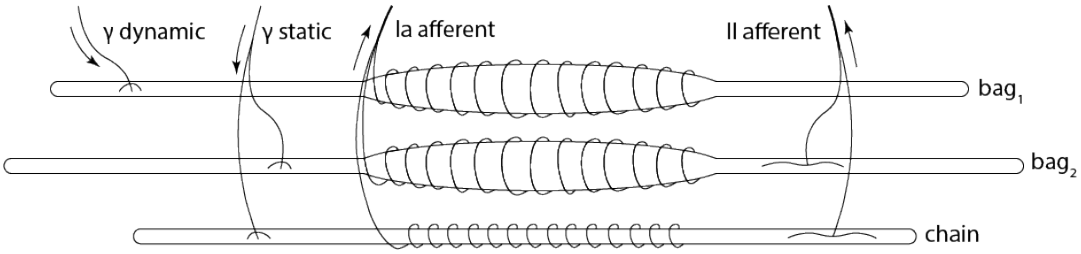


FIGURE 5.1: Biological model of the muscle spindle. The three intrafusal fiber types receive different fusimotor stimulations (static and dynamic) and produce primary (Ia) and secondary (II) afferent activities. Adapted from (Proske, 1997).

In (Mileusnic et al., 2006), the authors propose modelling all intrafusal fiber types with the same function, with different parameters based on physiology. This functions have two inputs: the fascicle length L (and its derivatives $\dot{L}$, $\ddot{L}$) and the relevant fusimotor activation level ($f_{dynamic}$, f_{static}). The fiber model consists of a sensory (SR) and polar (PR) regions, modeled as a pure elastic element and as a spring with a parallel active contractile element, respectively. From these models, given the inputs, one can compute the tension of the whole fiber T through a second-order differential equation. Once the tension is computed, the contribution of the fibers to the afferent rates can be obtained as follows:

$$rate_{bag_1}(t) = G \cdot \left[\frac{T(t)}{K^{SR}} - (L_N^{SR} - L_0^{SR}) \right] \tag{5.1}$$

$$rate_{bag_2}(t),\ rate_{chain}(t) = G \cdot \left\{ \begin{array}{l} X \cdot \frac{L_{sec}}{L_0^{SR}} \cdot \left[\frac{T(t)}{K^{SR}} - (L_N^{SR} - L_0^{SR}) \right] + \\ +(1-X) \frac{L_{sec}}{L_0^{SR}} \left(L(t) - \frac{T(t)}{K^{SR}} - L_0^{SR} - L_N^{PR} \right) \end{array} \right\} \tag{5.2}$$

A complete description of all the parameters found in Equations 5.1 to 5.8 and their values, which differ for every type of fiber, can be found in (Mileusnic et al., 2006), Table I. For a more detailed discussion of how the fibers are modeled, refer to (Mileusnic et al., 2006), from which Equations 5.1-5.4, 5.6-5.7 were adapted. The contributions are then combined to generate the firing rates of primary and secondary afferents:

$$rate_{II}(t) = rate_{bag_2}(t) + rate_{chain}(t) \tag{5.3}$$

$$rate_{Ia}(t) = \left\{ \begin{array}{ll} rate_{bag_1}(t) + S \cdot rate_{II}(t) & \text{if } rate_{bag_1}(t) > rate_{II}(t) \\ rate_{II}(t) + S \cdot rate_{bag_1}(t) & \text{if } rate_{bag_1}(t) < rate_{II}(t) \end{array} \right. \tag{5.4}$$

therefore computing the secondary afferent rate as the direct sum of bag_2 and chain activities, and the primary as a weighted sum between the three, which models the partial occlusion effect described in neuroscientific experiments. Such rates can be used to compute inter-spike time intervals to generate discrete spike events in a spiking neural simulator.

Because of the computation of $T(t)$ as a second-order differential equation, the original model is not well suited for the integration in applications where information regarding acceleration is unavailable or very noisy. This is because even a medium level of noise can lead to instability while performing a double integration.

To avoid this, the model was simplified by setting $\ddot{L} = 0$. The impact of this change is small, as the mean difference between the afferent firing rates with and without acceleration, computed on a Simulink implementation of the original model, was lower than 1% in most cases. This simplifies the tension equation for the polar and sensory regions, which can now be rewritten as follows:

$$T(t) = \begin{cases} K^{SR} \cdot \left(L(t) - L^{PR}(t) - L_0^{SR}\right) & \text{for sensory region} \\ \beta(t) \cdot C \cdot \left(L^{PR}(t) - R\right) \cdot \text{sign}\left(L^{P\dot{R}}(t)\right) \cdot \left|L^{P\dot{R}}(t)\right|^a + & \\ +K^{PR} \cdot \left(L^{PR}(t) - L_0^{PR}\right) + \Gamma(t) & \text{for polar region} \end{cases} \tag{5.5}$$

where β and Γ depend on the current fusimotor activation:

$$\beta(t) = \beta_0 + \beta_1 \cdot f_{dynamic}(t) + \beta_2 \cdot f_{static}(t) \tag{5.6}$$

$$\Gamma(t) = \Gamma_1 \cdot f_{dynamic}(t) + \Gamma_2 \cdot f_{static}(t) \tag{5.7}$$

Given that the tension of the two region of the fiber is the same, Equation 5.5 can be rearranged into a first-order differential equation:

$$\dot{T}(t) = \dot{L}(t) - signpow\left(\frac{T(t) - K^{PR} \cdot \left(L(t) - L_0^{SR} - \frac{T(t)}{K^{SR}} - L_0^{PR}\right) - \Gamma(t)}{\beta(t) \cdot C \cdot \left(L(t) - L_0^{SR} - \frac{T(t)}{K^{SR}} - R\right)}, a\right) \cdot K^{SR} \tag{5.8}$$

where

$$signpow(x, a) = \text{sign}(x) \cdot |x|^a \tag{5.9}$$

After this rearrangement we have a model, Equation 5.8, whose only inputs are L, $\dot{L}$, $f_{dynamic}$, and f_{static}. Information coming from actual sensors can thus be processed to generate L and $\dot{L}$, in units of the rest fascicle length L_0, to provide dynamic input to the model.

5.2.1 Spiking fusimotor input and parameter identification

In (Mileusnic et al., 2006), $f_{dynamic}$ and f_{static} activations with values between 0 and 1 are computed from the actual fusimotor frequencies $\gamma_{dynamic}$ and γ_{static} using a biochemical Hill-type equation from analogue values of the firing rates, in conjunction with low-pass filtering. However, this solution relies on the instantaneous firing rates of γ-motoneurons. Such rate, in a spiking neural network simulation, cannot be accurately computed without introducing delays, as using the inter-spike interval can lead to very noisy results in case of irregular spike trains, while averaging it over time bins would make the rate smoother but it would introduce delays. Therefore, this type of computation is not suitable to be integrated in a spiking neural network that simulates the activity of γ-motoneurons. In fact, the spindle model should receive spike events and integrate them to compute the activation levels of fusimotor activity. To do so, it was decided to employ a spike integration mechanism, adding spikes with an instantaneous response and exponential decay, similarly to an exponential synapse model. The actual spike response r was scaled with respect to the current value of the activation f to keep the results between 0 and 1:

$$\dot{f}(t) = \left(e^{-\frac{1}{\tau}} - 1\right) \cdot f(t) + r \cdot (1 - f(t)) \cdot u(t) \tag{5.10}$$

$$u(t) = \begin{cases} 1 & \text{if a spike is received at time } t \\ 0 & \text{if no spikes are received at time } t \end{cases} \tag{5.11}$$

where r is the maximum impulse response and τ is the decay time. In the reference model, the activation level for chain fibers was not implemented with a low-pass filter but as a direct saturation function of the static γ-motoneuron firing rate. However, as already stated, it is not possible to compute the instantaneous firing rate during spiking network simulations, so employed the same spike integration technique for the fusimotor activation of chain fibers. Another possible approach to spike integration is to employ α-shaped responses. This leads to similar results; however, such a spike integration technique cannot be implemented with the SpiNNaker APIs, as they currently only support exponential synapses.

Then, a model selection procedure was employed to select the maximum impulse response r and decay time τ for the spike integration mechanism (Equation 5.10). In the original model, the activation level of fusimotor activation under constant γ-motoneurons activity rose and stabilized at a certain maximum value. Therefore, a dataset of these maximum values was created, using the Hill-type activations, for different frequencies of motoneurons firing rates: $\{10, 50, 75, 100, 150\}$spikes/s. The model selection procedure was then performed on all the different combinations of values for r and τ ranging in $0.01 \leq r \leq 0.4$ and in $100ms \leq \tau \leq 500ms$, with discrete sampling. To evaluate the parameters, the spike integration was simulated at a given frequency for 3s, and the percentage error of the obtained maximum activation levels compared with those in the dataset (*magnitude error*) was computed. As it reported in (Mileusnic et al., 2006), fusimotor activation reaches 90% of the maximum value in 343ms for $f_{dynamic}$ and in 471ms for f_{static}. To match this property, another criterion evaluated was the obtained percentage at these specific points in time and computed the discrepancies with 90% (*shape error*). During the parameter identification procedure, input for the spike integration was simulated by generating spikes at fixed intervals, which produced a regular oscillation in the computed activations. To limit the effect of these oscillations on the selection procedure, errors were computed on a smoothed activation. For each parameter combination, the magnitude and shape errors were summed up and averaged on the different frequencies of the dataset. At the end of the procedure, these values had the smallest average percentage error (7%):

$$r_{dynamic} = 0.08, \qquad \tau_{dynamic} = 310ms$$
$$r_{static} = 0.09, \qquad \tau_{static} = 425ms$$

These values provide accurate results for the fusimotor activity ranging from 30spikes/s to 150spikes/s, while they tend to make the spike integration mechanism underestimate f for lower stimulation frequencies and to overestimate it for higher ones. For the static activation level of the chain fiber, f_{static} was scaled, relative to bag_2, by a factor equal to the mean scaling factor between the maximum levels of the original model at the same stimulation frequencies of the dataset. The averaged scaling factor was 0.829. A comparison between the original activations and those obtained with the spike integration procedure can be seen in Figure 5.2, where constant dynamic and static fusimotor activities ($\gamma_{dynamic} = 75$spikes/s and

$\gamma_{static} = 50$spikes/s) are converted into the corresponding fusimotor activation levels.

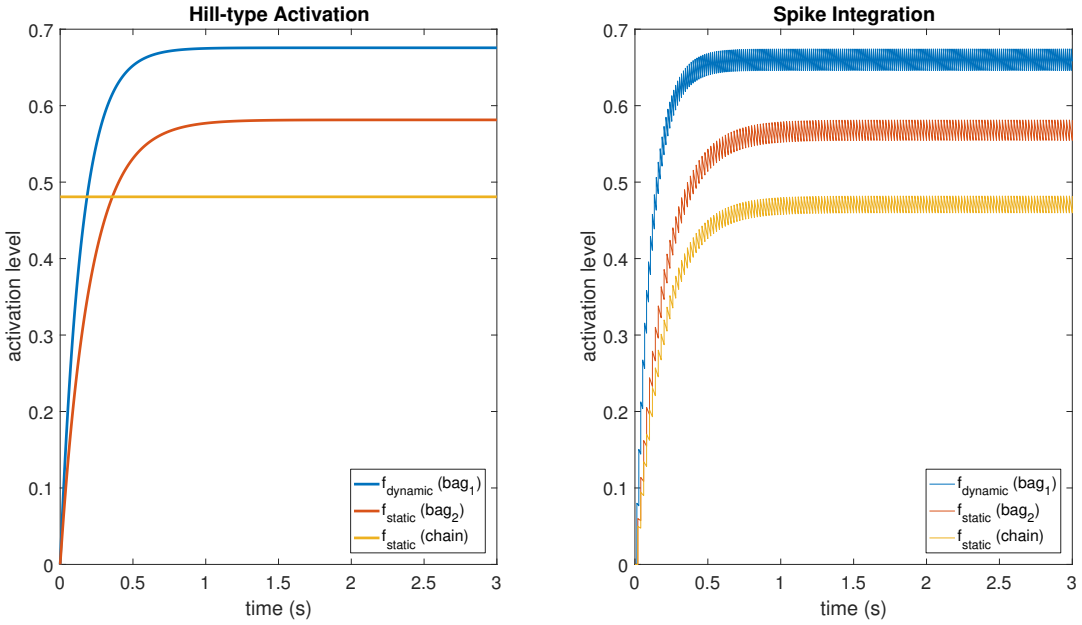


FIGURE 5.2: A comparison between activation levels obtained with the Hill-type equations (with low pass filtering) and those obtained with the spike integration procedure for a constant dynamic fusimotor stimulation of 75spikes/s and a static stimulation of 50spikes/s. The regular oscillation of the activation levels in the spike integration case is produced by the input spike trains, that were generated at fixed time intervals.

5.3 Spinal cord model

To develop the final model, an incremental approach was followed, starting from a circuit for a single *muscle*, adding heteronymous pathways from synergistic muscles in the same *group*, and finally inhibitory connections between *antagonistic* pairs (Figure 5.3).

5.3.1 Spinal neurons for a single muscle

For a single muscle, a network with muscle spindles providing Ia and II afferent fibres activity, a pool of α-motoneurons and excitatory II-interneurons was considered (Stienen et al., 2007; Moraud et al., 2016). Ia afferents directly provide excitatory inputs to the α-motoneurons (monosynaptic stretch reflex mechanism), while the II afferents output is mediated by a set of interneurons before reaching the α-motoneurons, creating a disynaptic reflex. The α-motoneurons receive an input from a spike generator that represent the descending motor command, while two additional spike sources simulate the activity of dynamic and static γ-motoneurons that regulates the spindle sensitivity. All other neurons are instead modelled as leaky integrate and fire neurons. Distribution of parameters for the α-motoneurons that influence the recruitment order and fibre strength (membrane capacitance, membrane

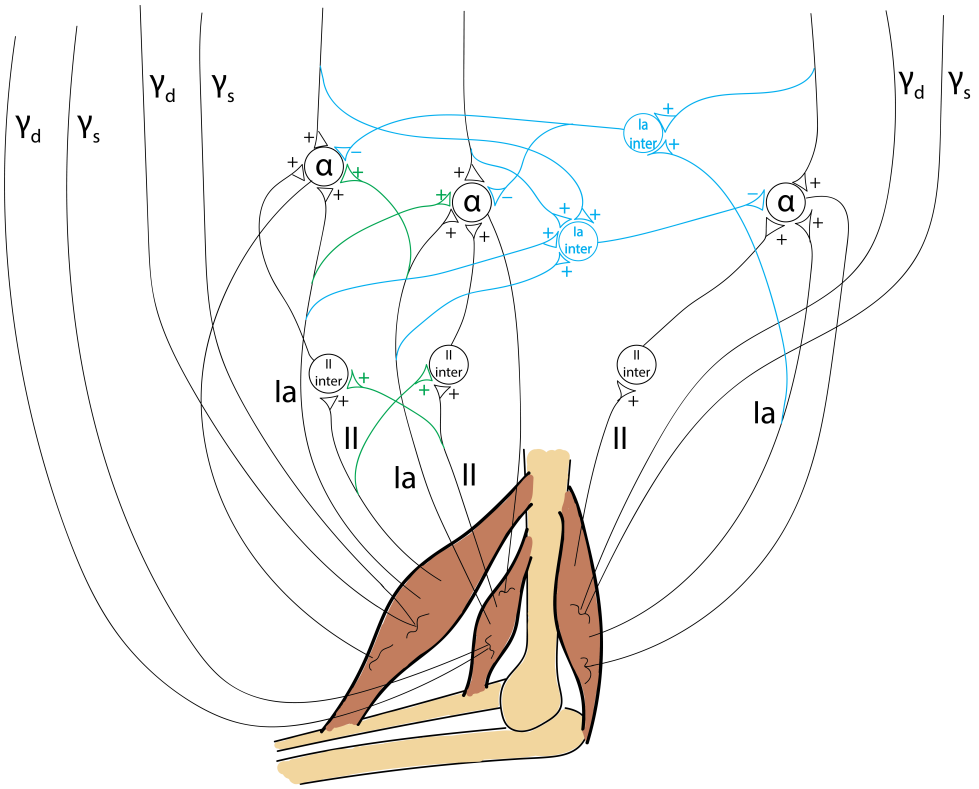


FIGURE 5.3: Spinal cord model for a muscle system with two synergistic muscles and one antagonist, such as the biceps-brachialis-triceps system. Heteronymous and inhibitory connections are highlighted with different colors.

time constant, maximum twitch force, time to peak force) are taken from (Sreenivasa, Ayusawa, and Nakamura, 2016):

$$D_i = [d_{max} - d_{min} \cdot \log(N - i)] \cdot D_{SF} \tag{5.12}$$
$$C_i = \pi D_i^2 \cdot c_{spf} \tag{5.13}$$
$$\tau_i = \tau_{max} - (D_i - \tau_{adj}) \cdot \tau_{slp} \tag{5.14}$$
$$F_i = [p_{max} - p_{min} \cdot \log(N - i)] \cdot F_{SF} \tag{5.15}$$
$$T_i = \left[s_{min} - \frac{s_{sl}}{N} i\right] \cdot T_{SF} + s_{min} \tag{5.16}$$

where i is the index of the α-motoneuron in the pool, N is the size of the pool and the others are free parameters that have to be adjusted for every muscle.

5.3.2 Muscle activation model

In order to compute the actual muscle activation from the motoneurons activity, a special spike integration unit that sums the fibres twitches was implemented. The spikes were integrated using the discrete time equations of (Cisi and Kohn, 2008) with a non-linear scaling factor from (Fuglevand, Winter, and Patla, 1993) that prevents the activation to grow indefinitely:

$$a_i(t) = 2e^{\frac{-\delta t}{T_i}} \cdot a_i(t-1) - e^{\frac{-2\delta t}{T_i}} \cdot a_i(t-2) + F_i \cdot g(t) \cdot \frac{\delta t^2}{T_i} e^{\frac{1-\delta t}{T_i}} \cdot u(t) \tag{5.17}$$

where δt is the integration time, and $u(t)$ and $g(t)$ are the spike function and the non-linear scaling, defined as:

$$u(t) = \begin{cases} 1 & \text{if a spike is received at } t \\ 0 & \text{if no spikes are received at } t \end{cases} \tag{5.18}$$

$$g(t) = \begin{cases} 1 & \text{if } T_i/ISI_i < 0.4 \\ \frac{1-e^{-2(T_i/ISI_I)^3}}{T_i/ISI_I} & \text{otherwise} \end{cases} \tag{5.19}$$

where ISI_i is the observed inter-spike interval of α-motoneuron i. Moreover, the activation can be scaled between 0 and 1 by dividing by the maximum theoretical value:

$$a_{i,max} = \lim_{\substack{t\to+\infty \\ ISI_i\to 0}} a_i(t) = F_i \frac{\frac{\delta t^3}{T_i^2}\left(1 - e^{-2\left(\frac{T_i}{\delta t}\right)^3}\right) \cdot e^{\left(1-\frac{\delta t}{T_i}\right)}}{1 - 2e^{-\frac{\delta t}{T_i}} + e^{-2\frac{\delta t}{T_i}}} \tag{5.20}$$

Therefore, the output of the twitch integration module is an activation value in $[0; 1]$ that is suitable for both muscle models, including Hill-type, and physical actuators. An example of twitch integration, with $T_i = 0.03s$, $F_i = 10mN$ and constant stimulation frequencies can be seen in Figure 5.4 where it can be observed that at low frequencies the individual twitches can still be seen, while at higher stimulation the twitches fuse into a tetanic contraction. Moreover, thanks to the non-linear scaling, the activation reaches a maximum value and higher stimulation frequencies do not produce any effect, in accordance with the contractile properties of real muscle fibres.

The motor pool and muscle spindles were actually subdivided into different populations connected to different bundles of the same muscle (e.g. short and long head of biceps). However, it was assumed that the same descending signals provide inputs to all bundles of the same muscle, and that the feedback connections involve all bundles, meaning for instance that Ia afferents of a bundle are connected the α-motoneuron pool of another one.

5.3.3 Network for synergistic and antagonistic muscles

In humans and other animals, a contribution to the stretch reflex of a muscle is also given by the afferent fibres of synergistic muscles, through heteronymous connections (Chalmers and Bawa, 1997). Such connections, from Ia afferents and II-interneurons of a muscle to the α-motoneurons of the other are less strong and less common, thus in the current implementation it was decided to limit them to only 60% of the total motoneuron population, maximum recorded value in humans, and to scale the weights by 0.6, compared to ones of the homonymous.

Finally, in order to implement the polysynaptic inhibition reflex, two populations

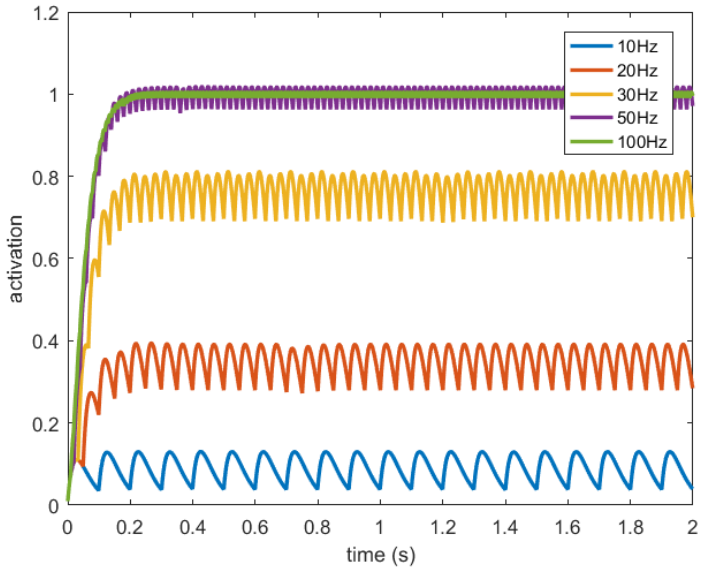


FIGURE 5.4: Twitch integration of constant stimulations with different frequencies, $T_i = 0.03s$, $F_i = 10mN$.

of Ia-interneurons were added to the network. Those receive inputs from all Ia afferents of synergistic muscles and provide inhibition to the α-motoneurons of the antagonistic muscles. Moreover, as the activation of a muscle should provoke an inhibition of its antagonist (Pierrot-Deseilligny and D. Burke, 2005), the Ia-interneurons also receive low-gain positive inputs from the corresponding descending pathways.

5.4 Implementation

The muscle spindle model was implemented into two commonly available simulation platforms: NEST and SpiNNaker. NEST is a point-neuron simulator for spiking neural networks with an extendible comprehensive library of different neuron and synapse models, including plasticity mechanisms, already described in Chapter 4 as it a fundamental part of the Neurorobotics Platform. SpiNNaker, however, offers a hardware platform with a massively parallel set of low-energy cores, and it focuses on real-time simulations. Currently, the library of neural models and synapses is limited, but the provided APIs ease the development of new ones. In the experiments, a SpiNN-5 board with 48 chips, each one with 18 cores, was employed.

For NEST, the implementation proceeded by using the provided APIs to create a new neural model that can be simulated alongside other neuron types and receive spike events. The fiber tension for each type of fiber was computed by performing a discrete fixed-step integration of Equation 5.8. Fusimotor activation was computed by the aforementioned spike integration (Equation 5.10). Spikes are transmitted to the spindle model through two different synapse types, representing dynamic and static efferent fibers, to separate the incoming spike events. After the computation of the Ia and II afferent rates, actual spikes are generated using a Poisson distribution for the inter-spike interval, employing the existing NEST utilities. Because in NEST simulation every neuron model can have only one output channel, a single spindle unit cannot produce both primary and secondary afferent activities. To overcome this limitation, a Boolean flag to the model that can be used to switch between the

two afferent types was used, that defaulted to primary. As a consequence, if one wants to simulate a certain number n of complete muscle spindles, he should create $2n$ units and set the appropriate flags for half of population. Another possibility, equivalent in terms of produced output and performances, would have been to develop two separate models for primary and secondary activity. However, it was decided to pursue the first strategy to provide a uniform interface with the SpiNNaker model, where the two choices are not equivalent, as described in the next paragraph. At the end of the implementation, the model was available both from the SLI interface and from PyNEST, and its parameters, such as the fiber length and speed, can be set using standard NEST calls. However, as with any other NEST model parameter, to set it to a new value, the simulation must be stopped and resumed after the change.

The same procedure was employed for the implementation on the SpiNNaker neuromorphic hardware, using a fixed-step integration of Equation 5.8 and spike integration for the fusimotor activation as in Equation 5.10. Spikes can be transmitted to this model via two custom synapse receptors, labeled *dynamic* and *static*. Because of the limitations of the ARM chips of the board, it was necessary to implement fixed-point arithmetic functions for the implementation of Equation 5.8, such as division and exponentiation. The fiber length and speed can be injected into a running simulation by using a customized version of a spike live injector model found in the SpiNNaker library. In this way, information can be sent without stopping the simulation, fulfilling the real-time constraint of the whole system. To maximize the performance of the model and ensure real-time execution of the simulation, in every population of spindles, which should be relative to a single muscle, the fiber tension was computed for a single spindle unit, comprising all fiber types. Even if this decreases the accuracy of the simulation, this does not compromise the plausibility of the model, as it is equivalent to assuming that the central nervous system provides the same fusimotor stimulation for all spindles in a muscle. After the computations, the resulting Ia and II afferent rates were employed to generate spikes with Poisson inter-spike intervals from all spindle units. The random process was emulated using an approximation of a homogeneous Poisson process (Heeger, 2000):

$$P\{\text{spike during } \delta t\} = rate \cdot \delta t \tag{5.21}$$

where δt is the duration of a simulation time step. This approach was suitable because of the fixed time bins (the simulation steps) and of the small δt (1ms). At the end of the implementation procedure, the model, developed in C, could be instantiated from the SpiNNaker PyNN frontend (Davison et al., 2008). As for the NEST implementation, a Boolean flag to switch between primary and secondary afferent activity was added. In this case, developing two different models for primary and secondary activity would have a negative impact, as SpiNNaker allows us to simulate only homogeneous populations on its cores. This implies that primary and secondary afferents spindle models, relative to the same muscle, would be split into two populations, thus duplicating the fiber tension computation and forcing the user to create additional customized spike injectors.

The muscle activation model was, for the time being, only implemented in NEST as a custom node able to integrate the twitches from a population of α-motoneurons. As such, the different twitches amplitudes and time constants can be set as parameters to the model.

Moreover, a generic implementation of the spinal circuitry was developed so that different muscle sets can be easily created by specifying parameters such as the

number of motoneurons, spindles and interneurons, values for Equations 5.12-5.16, and the antagonistic relationship between muscles. Moreover, the implementation allows to easily scale the number of total neurons by automatically adjusting the synaptic weights accordingly. This allows a faster testing phase in which the accuracy of the network is not severely compromised.

5.5 Results for the muscle spindle implementation

To assess the effectiveness of the proposed model, several experimental trials were performed. In particular, the NEST and SpiNNaker implementations were validated against a MATLAB Simulink simulation, comparable with the one performed in (Mileusnic et al., 2006). Then, the model was employed to translate sensory information from different simulated and physical robotics systems, proving the generality of the approach. In all the experiments, the parameter values were kept unchanged from the one reported in (Mileusnic et al., 2006), which are optimized on cat soleous muscle recordings. However, the same anatomical structure exists in all mammals; therefore, because of the accurate representation of the anatomy of the model by changing the parameter values one could in principle reproduce the different properties of the various muscles. In the implementations, the possibility to change such parameters was left open by providing the appropriate NEST and PyNN interfaces.

5.5.1 Validation

As the first step, a MATLAB Simulink model of Equations 5.1-5.11 was developed, whose results were directly, albeit empirically, comparable with those reported in (Mileusnic et al., 2006). Then, the results were compared, in terms of afferent rates, of the two different implementations, with the Simulink reference by executing the same tasks in terms of fiber stretch and fusimotor stimulations. After the execution, the spike trains were recorded, and the rate was computed by sorting them into bins of fixed time intervals (30ms) and counting them to compute the average rate for the bin.

In particular, a simple stretch was simulated, with the fiber length remaining constant at $0.95L_0$ for 1.1s, stretching at $0.11L_0$/s for 1.1s, and then remaining constant at $1.08L_0$ for the final 1.1s. The same stretch was repeated under different fusimotor drives ($\gamma_{dynamic}$ = 70spikes/s and γ_{static} = 70spikes/s). To provide the fusimotor stimulations in NEST and SpiNNaker simulations, Poisson spike generators were connected to the appropriate synapses types. Conversely, in the Simulink implementation, activities of γ-motoneurons were simulated by generating uniformly distributed spike trains. The number of muscle spindles simulated on NEST was 200, resulting in 400 simulated nodes, and 100 on SpiNNaker (200 nodes) because of memory per core limitations.

Results for this validation procedure can be found in Figure 5.5, where the response of the muscle spindle models, in terms of both computed spike rates and spike trains are shown. When no fusimotor stimulation is present, the spindle activity is null when the muscle is contracted; then, it starts to increase when the stretch begins and continues to rise as the stretch continues, and finally, it decreases and stabilizes at a certain level. Because there is no fusimotor stimulation, the activity of primary and secondary afferents is very similar. However, under a $\gamma_{dynamic}$ stimulation of 70spikes/s, Ia and II activities are radically different. In fact, primary afferents, the only ones affected by dynamic γ-motoneurons, have a greatly increased

response, especially during the elongation phase, showing an increased sensibility to stretch speed. In contrast, static γ-motoneurons provide an overall increase in sensitivity of both Ia and II afferents, providing sensory feedback even when the muscle is contracted, as shown for $\gamma_{static} = 70$spikes/s. From the results, we can confirm that the effects of γ-motoneuron stimulation during the simulations are in agreement with the spindle anatomy. Moreover, the results, in terms of computed spike rate, of the NEST and SpiNNaker implementations are very similar to the Simulink implementation, even if they are more noisy, thus proving their correctness. Finally, by comparing these results with those presented in (Mileusnic et al., 2006) for a similar trial, we can observe that the removal of the second-order terms and the spike integration did not change the results significantly, as the behaviors of the firing rate responses are preserved.

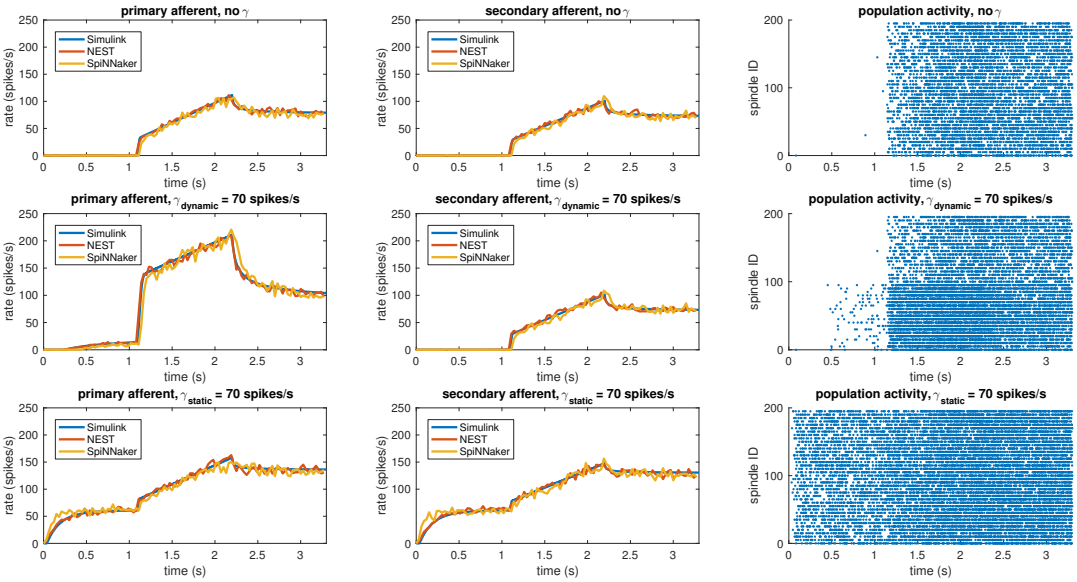


FIGURE 5.5: Comparison between the different implementations (Simulink, NEST, and SpiNNaker) for a stretching task with different fusimotor activities (no activity, dynamic at 70spikes/s, and static at 70spikes/s). The rows correspond to the different fusimotor activations. The first column shows the Ia afferent activity, in terms of spike rates, the second II afferent activity, and the third the raster plots of the neural population relative to the task, as produced by SpiNNaker. Spindles 0-99 simulate Ia activity, while spindles 100-199 simulate II activity. To improve visibility in the raster plots, only activity of 20% of the units of the spindle populations is displayed.

The mean simulation time for the NEST implementation was 7.51s on an i7-2760QM processor, implying a real-time factor of 0.44. The low real-time factor is due to the fact that the simulation must be stopped and restarted to set the values of L and $\dot{L}$. To compare the performance of the NEST model with those of other commonly used neural models, a continuous simulation of 100 primary afferents under constant dynamic fusimotor stimulation of 100spikes/s for 1 second was performed, with $L = 1$ and $\dot{L} = 0$, which resulted in an execution time of 0.48 seconds. By comparison, under the same stimulation, 100 leaky integrate and fire models had an execution time of 0.06s and 100 adaptive leaky integrate and fire models run for 0.81s. Therefore, the execution time of the NEST implementation falls between those of commonly used neural models. In contrast, simulations on SpiNNaker were able

to run in real time, and 200 spindle models (primary and secondary) could be simulated by a single core, using only 1% of the maximum theoretical capabilities of the hardware.

5.5.2 Sensory translation experiments

Once validated, the model can be used to translate proprioceptive feedback from robotic systems, both simulated and physical. In the tests, it was employed to convert information coming from motor encoders into afferent activity, but the same mechanism can be applied to translate proprioceptive information from more realistic biomechanical models, such as musculoskeletal simulations, or different sensors, such as physical stretch sensors.

First, the model was tested by embedding the NEST implementation in the Neurorobotics Platform. To do this, the NEST spindle model had to be integrated in the list of possible devices and then the transfer functions for the specific setups had to be developed. In particular, the spindle neural models was connected to two different robotic embodiments: an iCub robot and a simulated mouse body. In principle, every joint connecting two links can be considered actuated by an agonist-antagonist pair of muscles. Therefore, sensory information should be translated in terms of stretches of such muscles. This is demonstrated on a simulated iCub robot, where spindle model is employed to translate information received from the elbow encoder into afferent activities for an antagonistic pair of simulated muscles. The stretch and speed of the simulated muscles were computed geometrically, as a function of the encoder values $(\theta(t), \dot{\theta}(t))$ and of kinematic parameters of the links (arm and forearm, cf. Figure 5.6). Assuming, for simplicity, that the muscles are attached in the middle point of the two links and that $0 \leq \theta(t) \leq 2\pi$, the stretch and speed for the agonist and antagonist muscles can be computed as follows:

$$L_{ago}(t) = \sqrt{\frac{l_1^2}{4} + \frac{l_2^2}{4} - \frac{l_1 l_2}{2}\cos(\theta(t))} \tag{5.22}$$

$$\dot{L_{ago}}(t) = \frac{\frac{l_1 l_2}{4}\sin(\theta(t)) \cdot \dot{\theta}(t)}{L_{ago}(t)} \tag{5.23}$$

$$L_{ant}(t) = \frac{l_1}{2} + \frac{l_2}{2} + 2s \cdot \sin\left(\frac{\pi - \theta(t)}{2}\right) \tag{5.24}$$

$$\dot{L_{ant}}(t) = -2s \cdot \cos\left(\frac{\pi - \theta(t)}{2}\right) \cdot \dot{\theta}(t) \tag{5.25}$$

This system of equations can be employed on any two-link, asymmetric system (e.g., thigh-leg), by simply changing the kinematic parameters of the links or the attachment point of the two muscles.

To test it, a simulated iCub robot was placed inside a virtual room, where the experiment took place. A sinusoidal motion with a peak-to-peak amplitude of 45 degrees and a frequency of 0.2Hz was given to the elbow motor to simulate a rhythmic co-activation of the two muscles and an alternation of stretching and elongating of the corresponding spindles. The motion was centered on what was considered the resting position for the computation of L_0, 125degrees, an angle where none of the two simulated muscles is completely stretched (Figure 5.6). To stress the difference between primary and secondary afferent endings, fusimotor activation was set to $\gamma_{dynamic} = 70$spikes/s during the trial. The activity of the spindles during this

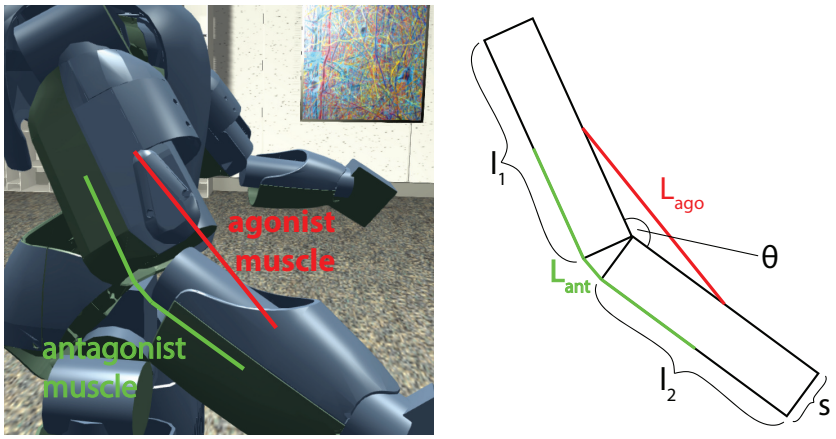


FIGURE 5.6: Simulated agonist-antagonist muscle pair for a two-link system and its application to the elbow joint of the iCub robot (left). The length of the two muscles can be expressed as a function of the kinematic parameters of arm and forearm and of the current joint angle.

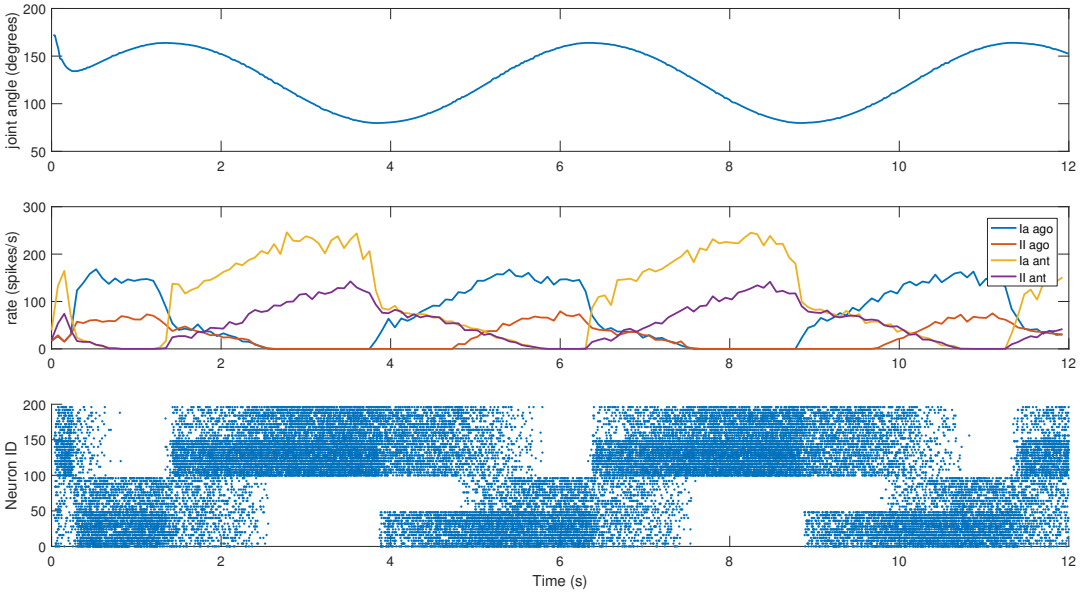


FIGURE 5.7: Afferent activity for the agonist-antagonist pair during a sinusoidal movement of the iCub elbow (top). Computed rates for Ia and II activities are shown in the middle, and the raster plot for the two muscle populations is on the bottom. To improve visibility in the raster plot, only activity of 25% of the units of the spindle populations is displayed.

trial are shown in Figure 5.7. It can be observed that the activity is not symmetric, as expected from the geometrical translation model employed, but that afferent rates values correctly alternate between the two simulated muscles, following the elbow motion. Moreover, Ia activity is higher than II, thanks to the dynamic fusimotor drive. Because no static fusimotor activity is present, when the fibers are contracted, there is no activity to be recorded. The computational burden of the simulation of 200 spindle models, combined with the physical simulation, resulted in a real-time

factor of the whole coordinated simulations of 0.16.

To consider a different link system, the spindle model was connected to simulate the afferent activities of muscles connected to a three-link kinematic chain, the shoulder-neck-head link system of a simulated mouse body, inside the Neurobotics Platform. The model consists of a rigid skeleton actuated by rotational joints, covered with deformable skin. The subset of interest of the skeletal model can be seen in Figure 5.8, where the relevant kinematic parameters are shown.

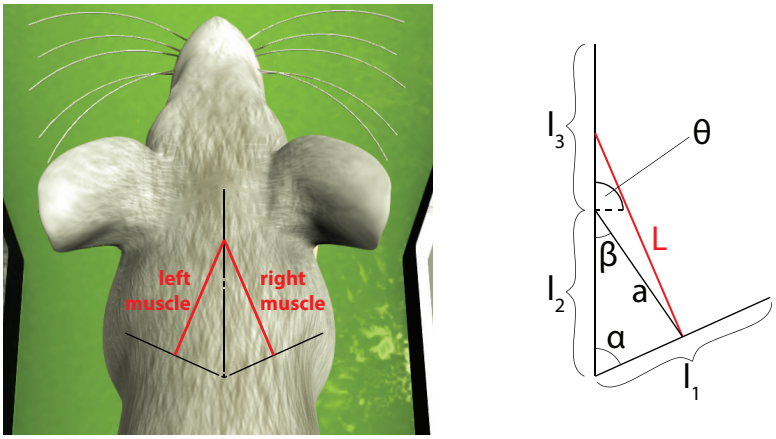


FIGURE 5.8: Simulated muscles for a symmetric three-link system and its application to the virtual mouse neck. The length of the muscle can be expressed as a function of the kinematic parameters of the three links and of the current two joint angles between them.

Compared to the previous case, this link system is symmetric, so the same equations can be used to compute $L(t)$ and $\dot{L}(t)$ by just changing the sign of $\theta(t)$ and $\dot{\theta}(t)$. Moreover, in this case, the muscle is attached to the first and third links. To simplify the equations, it was assumed the angle between the first two links (α) to be constant. However, small modifications are needed to consider it variable in time. The length and speed of the muscle spindles can be computed using the following equations:

$$a = \sqrt{\frac{l_1^2}{2} + l_2^2 - l_1 \cdot l_2 \cdot \cos(\alpha)} \tag{5.26}$$

$$\beta = \arccos\left(\frac{a^2 + l_2^2 - \left(\frac{l_1}{2}\right)^2}{2 \cdot a \cdot l_2}\right) \tag{5.27}$$

$$L(t) = \sqrt{\frac{l_3^2}{2} + a^2 - a \cdot l_3 \cdot \cos\left(\theta(t) + \frac{\pi}{2} - \beta\right)} \tag{5.28}$$

$$\dot{L}(t) = \frac{\frac{a \cdot l_3}{2} \sin\left(\theta(t) + \frac{\pi}{2} - \beta\right) \cdot \dot{\theta}(t)}{L(t)} \tag{5.29}$$

For this experiment, the mouse head was not moved directly, but relied on an existing setup where the mouse moved its head as part of a Braitenberg-like experiment: the mouse is placed in a Y-maze with two displays placed at the end of the corridors, and the mouse should look at the red one, away from the blue one, using

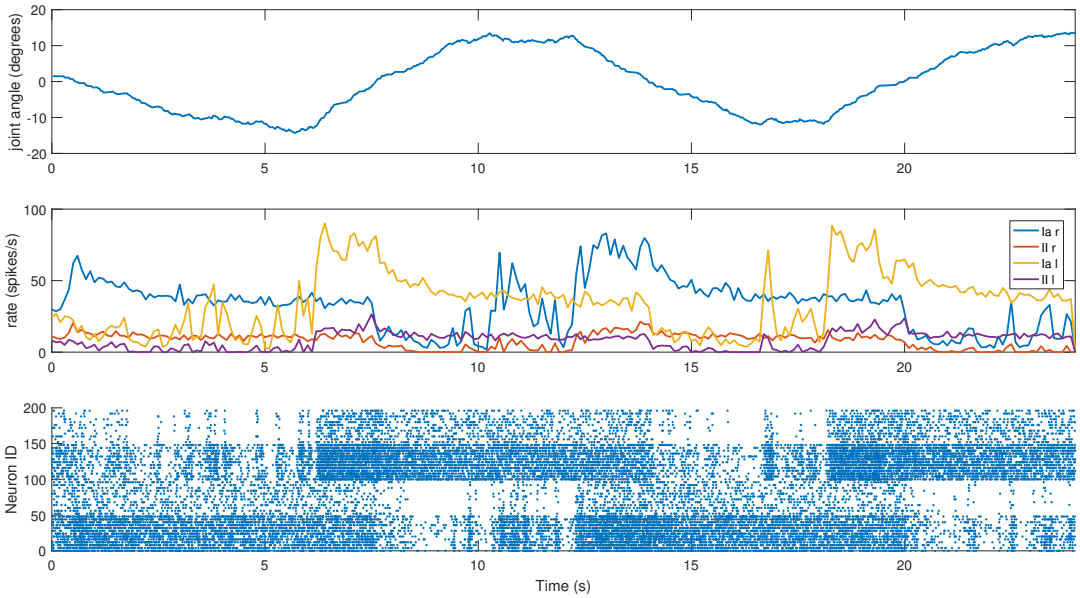


FIGURE 5.9: Afferent activity for the neck muscle pair during a screen viewing experiment. The head motion during the trial is on top, computed rates for Ia and II activities are shown in the middle, and the raster plot for the two muscle populations is on the bottom. To improve visibility in the raster plot, only activity of 25% of the units of the spindle populations is displayed.

a trivial neural model. The two screens switched states every 6 seconds, and the motion relies only on visual information, particularly on the percentage of red pixels in the camera image. The fact that the relevant joint is not explicitly controlled provides a more realistic scenario. The spindle activities was recorded during such a trial, again providing dynamic fusimotor drive ($\gamma_{dynamic} = 70$spikes/s) to enhance the differences between Ia and II activity. Results for the experiment can be found in Figure 5.9, where it can be observed that the activities of the spindles of the two muscles are symmetric and out of phase, in correspondence with the head motion. The controller does not provide smooth movements of the head, so the activity of the spindle models is noisy, especially in the presence of momentary and sudden changes in the fiber stretch speed. However, the activity of the two muscles and of the two different afferent types are clearly distinguishable from one another. It can be noticed that, because of the low range of motion, the firing rate of II afferents is very low. In addition, the stretching speeds are very low, but rates of the Ia afferents are still high thanks to the dynamic fusimotor activity. Finally, activity for contracted spindles is close to null, except for activity generated by the noisy input. In this case, the real-time factor of the coordinated simulation was 0.17.

To show the real-time capabilities of the SpiNNaker implementation, the spindle model was employed on a physical robotic platform. A three-link system was considered as starting from the shoulder link up to the head of an iCub robot, actuated by the neck roll joint. Because the the link system has the same structure as the previous case, the translation could be performed by using Equations 5.26-5.29, changing the kinematic parameters to match the iCub kinematics (Figure 5.10).

To actually send the translated encoder values to SpiNNaker, as well as to retrieve live spiking activity from the simulation, a proper real-time data exchange middleware was developed in C++. The joint speed cannot be directly retrieved

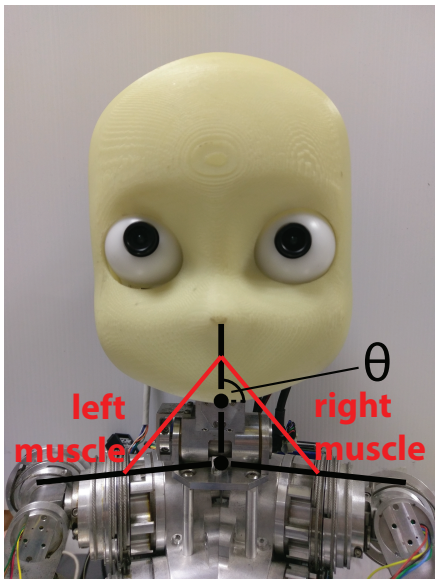


FIGURE 5.10: Simulated muscles for the iCub neck roll using the three-link system previously described for the mouse head.

from the motor, so it was necessary to compute it. To test the robustness of the spindle model against input noise, a simple single-step differentiation was employed. The neck roll joint was then moved in a sinusoidal fashion, with a peak-to-peak amplitude of 30degrees and a frequency of 0.5Hz, but the maximum stretch was maintained for 1 second on every side, by keeping the head still when it reached the maximum range of motion, resulting in a motion with a period of 4s. During this trial, fusimotor activity was kept at $\gamma_{dynamic} = 80$spikes/s and $\gamma_{static} = 40$spikes/s. Results for this trial can be found in Figure 5.11. As expected, the activities of the two sets of spindles are symmetric, out of phase from each other, and in sync with the motion. It is interesting to notice how Ia and II activities for the same muscle differ only during the actual motion part, while they tend to be almost the same when the head is still. Therefore, a model of the central nervous system, by properly activating γ-motoneurons, could really be able to discriminate between motion and different stretch levels. Moreover, thanks to static fusimotor activation, the spindles are overall more sensitive, and even when they are contracted, some activity is present. For the simulation of this model, to simulate the 200 spindles, divided into two neural populations, two cores of the SpiNN-5 board were used and thus only 0.2% of its maximum theoretical capabilities.

These results validated the neuromorphic muscle spindle model in isolation. After this phase this model was used as a building block to implement the more complex spinal cord model.

5.6 Results for the spinal cord model

The spinal cord model was then embedded in closed loop simulations with different embodiments: two musculoskeletal systems (human and mouse) and a humanoid robotic platform.

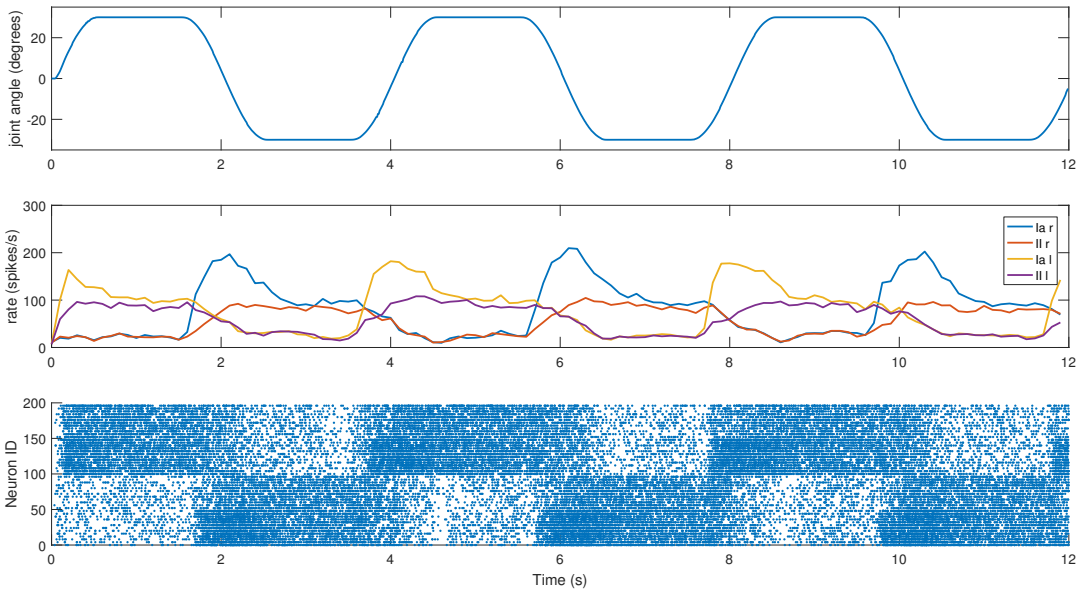


FIGURE 5.11: Afferent activity for the neck muscle pair simulated on the iCub robotic platform. The head motion during the trial is on top, computed rates for Ia and II activities are shown in the middle, and the raster plot for the two muscle populations is on the bottom. To improve visibility in the raster plot, only activity of 25% of the units of the spindle populations is displayed.

5.6.1 Stretch reflex simulation

As a first validation step, a stretch reflex experiment similar to the one presented in (Sreenivasa, Ayusawa, and Nakamura, 2016) was replicated. In (Sreenivasa, Ayusawa, and Nakamura, 2016), a muscular model composed of biceps and triceps (with 2 and 3 fibre bundles respectively) was employed. The same model, enriched with the brachialis synergistic muscle, was used in this case. The musculoskeletal simulation was performed using the dynamic simulator sDims (Nakamura et al., 2005). Figure 5.12 shows a 3D visualization of the muscle model considered.

The closed loop connections and synchronization between NEST and sDims was performed using a custom framework, that executed a parallel simulation step of 5ms before any data exchange. While the same parameters for the α-motoneurons and the same number of spindles were employed, the number of interneurons were chosen by scaling those found in (Bashor, 1998). Population sizes for the brachialis muscle were adapted from (Takeichi, 2015). A summary of the population sizes can be found in Table 5.1.

TABLE 5.1: Population sizes for the spinal cord circuit for the biceps-brachialis-triceps system. * indicates a shared population between biceps and brachialis.

muscle	α	spindles	II-inter	Ia-inter
biceps	774	320	897	1821*
brachialis	681	256	789	1821*
triceps	1571	520	1821	1687

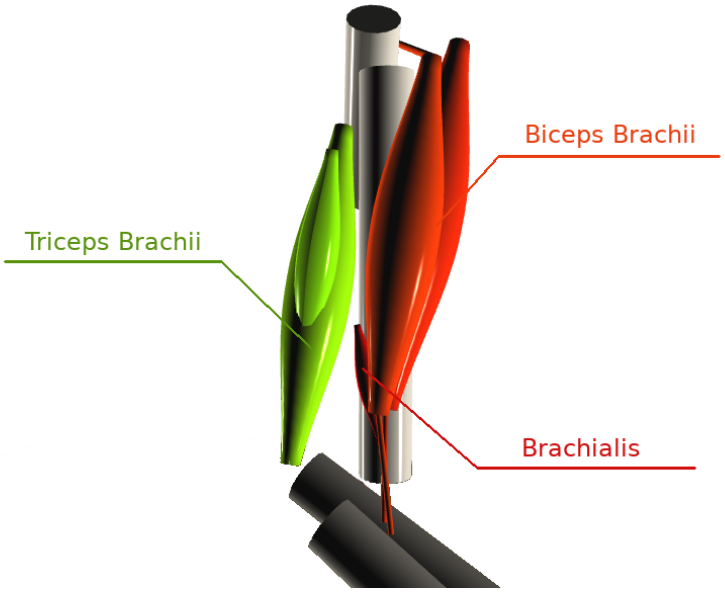


FIGURE 5.12: The muscle system considered as represented by the dynamic simulator sDims, with the corresponding skeletal structure.

Muscle forces for the musculoskeletal simulation are computed from the activations employing a Hill-type muscle model that computes the total force as the sum between active and passive elements (Sreenivasa, Ayusawa, and Nakamura, 2016; Stroeve, 1999; Buchanan et al., 2004):

$$F(t) = F_{max}\left(f_v(L(\dot{t}))f_l(L(t))a(t) + f_p(L(t))\right) \tag{5.30}$$

where F_{max} is the absolute maximum force of the muscle, $L(t)$ and $L(\dot{t})$ are the current muscle length and stretch speed and f_v, f_l, f_p are the components proportional to muscle stretch speed, length and passive elasticity, defined as

$$f_v(\dot{L}) = \begin{cases} 0 & \text{if } \dot{L} < V_{max} \\ \frac{V_{sh}(V_{max}+\dot{L})}{V_{sh}(V_{max}-\dot{L})} & \text{if } V_{max} \leq \dot{L} < 0 \\ \frac{V_{sh}V_{shl}V_{max}+V_{ml}\dot{L}}{V_{sh}V_{shl}V_{max}+\dot{L}} & \text{if } \dot{L} \geq 0 \end{cases} \tag{5.31}$$

$$f_l(L) = e^{-\left(\frac{L-L_0}{L_{cesh}}\right)^2} \tag{5.32}$$

$$f_p(L) = \frac{e^{10\left(\frac{L}{L_0}-1\right)}}{e^5} \tag{5.33}$$

where L_0 is the muscle rest length and the others are constant parameters described in (Stroeve, 1999). An example of the resultant force for an activation level of 1 and $F_{max} = 1$ can be seen in Figure 5.13.

In the experimental trial, a descending signal was given to all muscles so that they reached an activation level of around 0.25. After 0.5 seconds, a stretch on the biceps and brachialis muscles was simulated by artificially lengthening the muscle for 100ms and shortening again for 100ms, as if it were hit by a stretch hammer used in clinical trial. In Figure 5.14, by comparing the top row, where the reflex is disabled,

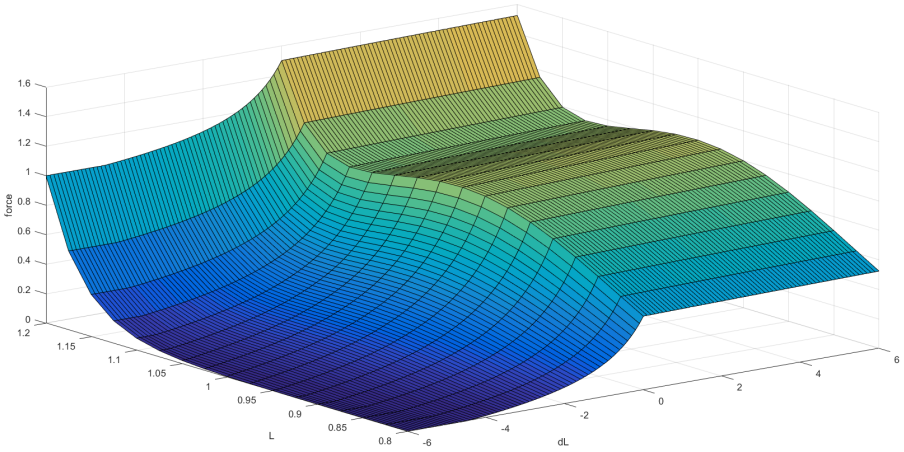


FIGURE 5.13: Force generated by the Hill-type muscle model for $a = 1, F_{max} = 1$.

with the middle row where is enabled, it can be observed that all reflexes are properly elicited. In particular, after 500ms, when the lengthening occurs, activity of the muscle spindles of the biceps and brachialis increase. As a consequence, the stretch reflex mechanism is activated and the firing rate of corresponding II-interneurons and α-motoneurons rises. Moreover, activity of the α-motoneurons of the triceps decrease due to the inhibitory polysynaptic pathway. If the same experiment is performed while providing dynamic fusimotor drive to the muscle spindles, we can observe that the response is a sharper reflex, that is with increased neural activity but less duration (Figure 5.14, last row).

5.6.2 Control of a robotic platform

To explore the possibility of using this kind of biologically inspired networks for the control of robotic platforms, the neuro-musculoskeletal described before was connected to a simulated robotic platform. A model of the iCub humanoid robot for the physical simulator Gazebo was modified in order to receive torque commands instead of position ones for the elbow joint. Changing the physical model from human to robotic was almost seamless, as the spinal network was kept the same as before, while the musculoskeletal system was just scaled to match the size of robot links. The dynamic simulator sDims was then used, at each simulation step, to compute the resulting torque at the elbow joint, given the current muscle forces. Such torque was then given to the robot as a motor command. After the physical simulation step, the elbow encoder values (angle, speed, acceleration) were given back to the sDims model to compute the new muscle lengths, closing the loop (Figure 5.15).

As a test of how the spinal cord circuitry can affect the behaviour of the control, a test in which a constant control signal was given to the α-motoneurons and a perturbation was applied to the robot forearm was performed. In particular, a weight was dropped on the forearm of the robot provoking a sudden change in the lengths and stretch speeds of biceps and brachialis muscles, eliciting a stretch reflex response. Results for this trials are shown in Figure 5.16, where it can be observed that, while

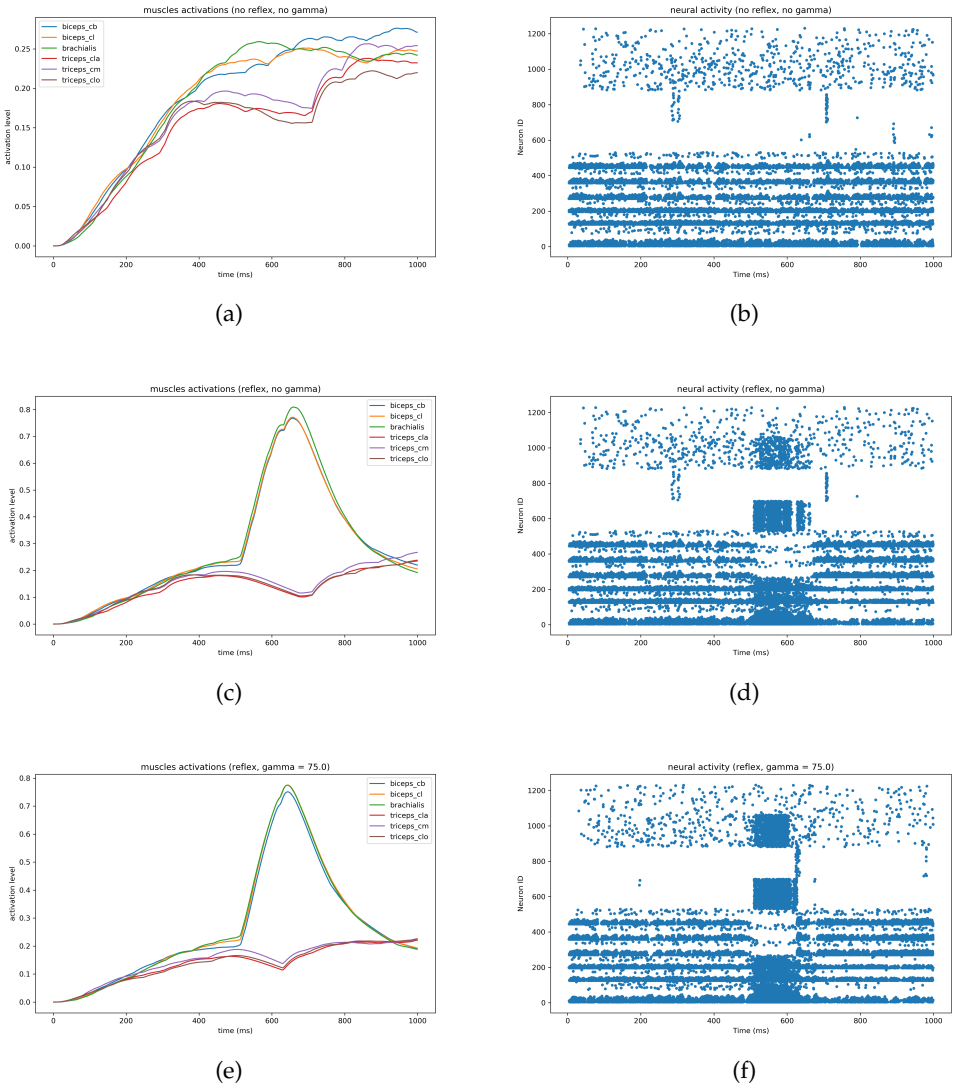


FIGURE 5.14: Simulated stretch reflex with a human musculoskeletal model. In the raster plots, only 10% of the whole populations are shown.

the controller without the reflex enabled is still able to recover from the perturbation, when the reflex is enabled, there is a peak in the joint torque in response to the perturbation that leads to a faster recovery.

5.6.3 Mouse musculoskeletal model

Finally, the spinal cord model was tested on a different musculoskeletal embodiment: a mouse forearm model, simulated with the OpenSim dynamic simulator (Delp et al., 2007) in a local install of the Neurorobotics Platform (Figure 5.17). The model includes three antagonistic pairs of muscles for three degrees of freedom. Therefore, a spinal cord model for such a pair was created and replicated three times. Parameters for the model were taken from (Moraud et al., 2016). For this trial there was no need to manually compute the Hill model, as the simulated muscle actuators already expected an input between 0 and 1. While no quantitative results have been gathered yet, the first tests conducted confirm that it is possible to reach different

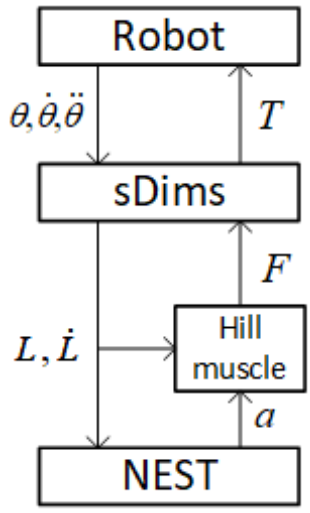


FIGURE 5.15: Closed loop architecture with NEST, sDims and the simulated robot.

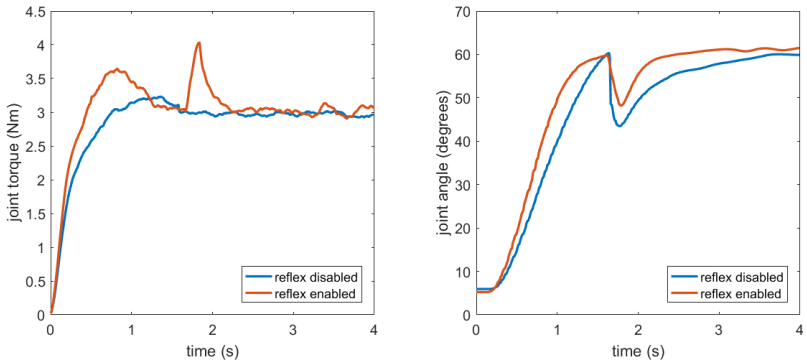


FIGURE 5.16: Comparison between perturbation responses without and with the stretch reflex enabled. The joint torque is displayed on the left, while the joint angle is on the right.

limb configurations, with different joint trajectories, in terms of speeds and accelerations, depending on the descending signals, and that the control is stable. This proves that the generality of the spinal cord model make it applicable for different embodiments.

5.7 Conclusions

In this chapter, a comprehensive model of spinal cord was developed and integrated in closed loop simulations with different embodiments, both musculoskeletric and robotic, as a first step towards more complex biologically inspired controllers.

The spinal cord circuit make use of a neuromorphic muscle spindle model able to produce biologically-realistic firing activity. The mechanism, that emulates the dynamic of muscle spindles under stretch and γ-motoneuron activation, was implemented on two different simulators: NEST, a commonly employed spiking neural network simulator, and SpiNNaker, a neuromorphic hardware platform. The model

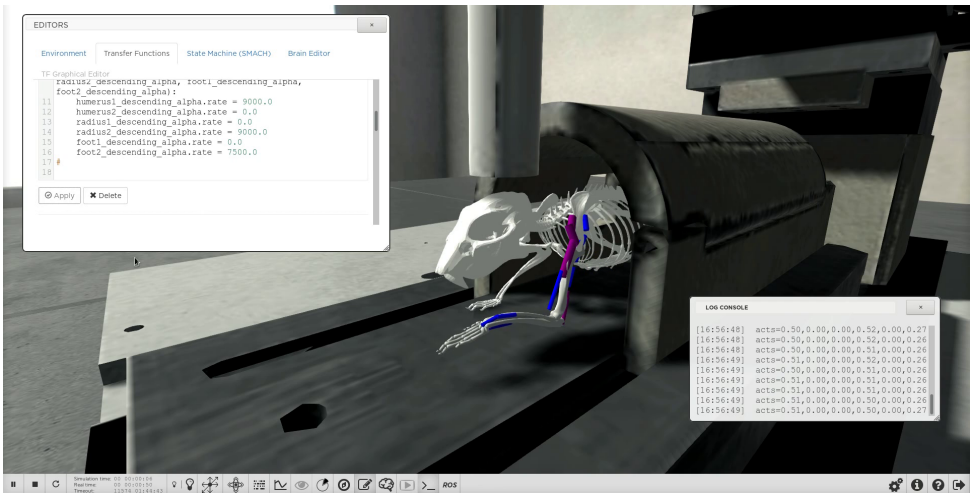


FIGURE 5.17: The mouse musculoskeletal model in the Neurorobotics Platform, controlled through the spinal cord model.

was obtained by modifying an already existing muscle spindle model (Mileusnic et al., 2006) in order to take perform spike integration to compute the fusimotor drives and to simplify the computation.

Results show that the spinal circuitry is capable of delivering appropriate motor commands to the agent, and that the spinal reflexes work properly. The model is also highly configurable and can be easily adapted to different scenarios.

Preliminary tests confirm that it is in principle possible to have a stable control with such a model, by giving appropriate descending inputs to α and γ-motoneurons. Therefore, the research should proceed into developing higher-level controllers that mimic functionalities of the sensory-motor cortex.

Finally, it has been shown that this model of control can be employed, without much additional effort, on robot platforms. This opens the possibility of creating neuro-controllers that can be tuned on human musculoskeletal recordings and ported with just a few adjustments to robotic platforms. This porting should be in principle seamless on robots with muscle-like actuation mechanisms, and can be greatly beneficial, as no effectively proven methods for controlling such robots exist yet. However, the feasibility of this approach must be proved first on real robotic platforms in real-time scenarios.

Chapter 6

Concluding remarks

This thesis presented some advancements in the field of humanoid robotics and in particular to the area of brain-inspired control. Motivated by the inability of state-of-the-art robots to perform realistic tasks in unstructured scenarios due to their lack of adaptability, this work make some steps towards overcoming these limitation. By employing neuro-controllers with artificial neural networks models able to exploit prediction to automatically adapt the behaviour of the robot were developed. Moreover, some foundational work for the integration of more sophisticated spiking neural networks was laid out.

6.1 Summary of the contributions

The major contributions can be summarized as follows:

- **An adaptive visual pursuit model** (Chapter 2):
  A novel smooth pursuit controller was developed and implemented. The model is able to combine sensory and predictive information through a switching mechanism to achieve zero lag tracking when the target motion is predictable while automatically adapting to changes in the target dynamics. The switching mechanisms enables the system to follow the target even when the trajectory is not predictable, with some delay. The controller was tested both in simulation and on a physical robotic platform.

- **A complete gaze stabilization system** (Chapter 3):
  By integrating different reflexive behaviours, a comprehensive gaze stabilization model was developed and implemented. The system includes the vestibulocollic reflex, the vestibulo-ocular reflex and the opto-kinetic reflex, and is able to move both eye and head joints in coordination, with the aim of better stabilizing the gaze. Every reflex make use of sensory-motor anticipation through cerebellar internal models and the system is able to automatically adapt to different disturbance dynamics without significantly changing the control parameters. The controller is also capable of offline learning. The system was tested in simulation and on physical robots: with a oscillating platform providing periodic disturbances and during biped locomotion.

- **A syncing mechanism between neural and robotic simulations** (Chapter 4):
  As a contribution to the development of the Neurorobotics Platform of the Human Brain Project, the synchronization mechanism between the physical and neural simulations was designed and implemented, as well as the proper

interfaces towards the robotic middleware. This contributed to the public release of this piece of software that grants neuroscientist and roboticists working with spiking neural networks and robotic systems an integrated environment where to test their models. Furthermore, preliminary validation of the system through simple yet complete neuro-controllers was provided.

- **A comprehensive spinal cord network** (Chapter 5):
  As a step towards a generic intermediate level between robotic systems and detailed brain simulations, a fairly complete spinal cord model was developed. The model includes type Ia and II sensory afferents from muscle spindles, α-motoneuron pools, a model of muscle twitches integration and the synaptic connections enabling mono and polysynaptic reflexes. The model was tested with both robotic and musculoskeletal embodiments.

6.2 Perspectives

To conclude this thesis, future research steps and possible directions to follow are discussed.

One short term results could be the integration of the smooth pursuit model with the gaze stabilization system, to achieve a more complete and coordinated control of the gaze. Wile this could be seen as trivial at a first glance, it would actually mean the development of an integration mechanism capable of coordinating the two that employs a set of inhibitory signals suppressing the stabilization reflexes during voluntary pursuit movements. Thus, this will require embedding more cognitive mechanisms in the system. An even more complex system could be created by integrating more eye movements such as saccades.

A more challenging step would be to develop more biologically realistic models of pursuit and stabilization using spiking neural networks. This would require translation mechanism such as the muscle spindle and muscle activation models described in Chapter 5 also for the vestibular system and detailed models of involved brain areas, some of which already exists (i.e. cerebellar microcircuit). The outcome of this work could be a more realistic system that mimics human behaviour just by means of the properties of the subsystems that constitute it and not because some adjustments were added to reproduce the observed behaviour. For instance, in a biomimetic gaze stabilization system, the fact that the vestibulocollic reflex is much slower than the vestibulo-ocular reflex, should be intrinsically modelled by the lower number of synapse of the reflex arc in the second case, without the need to artificially add delays to reproduce the observed behaviour.

Another challenging objective would be to give meaningful inputs to the spinal cord model via a spiking neural network implementing sensory motor maps of the cerebral cortex relative to arm coordination and control, in order to perform tasks such as reaching and grasping. This would involve at least receiving and integrating proprioceptive and exteroceptive sensory informations in a model of the somatosensory cortex and a model of the motor cortex for the movement generation and execution. Such a system could automatically provide human-like movement execution and, more importantly, would be directly usable on non-conventional robotic platforms such as those with tendon-driven or muscle-like actuation mechanisms. The architecture could then be incrementally improved with cerebellar model for smooth and well timed movement execution as well as basal ganglia models for action selection.

A different research direction that is already ongoing as part of the Human Brain Project is to employ models such as the retina simulator presented in Chapter 4 or the spinal cord not to implement neuro-controllers, but to reproduce neuroscientific experiments. In particular, the spinal cord model will be employed to simulate a stroke rehabilitation experiment on mice in conjunction with a data driven reconstruction of the whole mouse brain, in the Neurorobotics Platform. This should provide a more robust assessment of the validity of the model. If successful, reproducing real experiments in simulated environments could be disruptive and greatly fasten neuroscientific research.

Bibliography

Albus, J. S. (1971). "A theory of cerebellar function". In: *Mathematical Biosciences* 10.1-2, pp. 25–61.

Allard, J., S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni (2007). "Sofa-an open source framework for medical simulation". In: *MMVR 15-Medicine Meets Virtual Reality*. Vol. 125. IOP Press, pp. 13–18.

Asuni, G., G. Teti, C. Laschi, E. Guglielmelli, and P. Dario (2005). "A robotic head neuro-controller based on biologically-inspired neural models". In: *Proc. IEEE International Conference on Robotics and Automation (ICRA'05)*, pp. 2362–2367.

Banks, R. (1994). "The motor innervation of mammalian muscle spindles". In: *Progress in neurobiology* 43.4, pp. 323–362.

Barnes, G. (1993). "Visual-vestibular interaction in the control of head and eye movement: the role of visual feedback and predictive mechanisms". In: *Progress in Neurobiology* 41.4, pp. 435–472.

Barnes, G. and P. Asselman (1991). "The mechanism of prediction in human smooth pursuit eye movements." In: *The Journal of Physiology* 439.1, pp. 439–461.

Bashor, D. P. (1998). "A large-scale model of some spinal reflex circuits". In: *Biological cybernetics* 78.2, pp. 147–157.

Bäuml, B., O. Birbach, T. Wimböck, U. Frese, A. Dietrich, and G. Hirzinger (2011). "Catching flying balls with a mobile humanoid: System overview and design considerations". In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*. IEEE, pp. 513–520.

Beira, R., M. Lopes, M. Praga, J. Santos-Victor, A. Bernardino, G. Metta, F. Becchi, and R. Saltarén (2006). "Design of the robot-cub (iCub) head". In: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*. IEEE, pp. 94–100.

Benefiel, A. C. and W. T. Greenough (1998). "Effects of Experience and Environment on the Developing and Mature Brain: Implications for Laboratory Animal Housing". In: *ILAR Journal* 39.1, pp. 5–11. eprint: http://ilarjournal.oxfordjournals.org/content/39/1/5.full.pdf+html.

Bergamini, E., G. Ligorio, A. Summa, G. Vannozzi, A. Cappozzo, and A. Sabatini (2014). "Estimating orientation using magnetic and inertial sensors and different sensor fusion approaches: Accuracy assessment in manual and locomotion tasks". In: *Sensors (Switzerland)* 14.10, pp. 18625–18649.

Berthoz, A. (2002). "The brain's sense of movement: Perspectives in Cognitive Neuroscience". In: *USA: Harvard Univ Pr*.

Bloomberg, J., M. Reschke, W. Huebner, and B. Peters (1992). "The effects of target distance on eye and head movement during locomotion". In: *Annals of the New York Academy of Sciences* 656.1, pp. 699–707.

Bohren, J. and S. Cousins (Dec. 2010). "The SMACH High-Level Executive [ROS News]". In: *Robotics Automation Magazine, IEEE* 17.4, pp. 18–20.

Bouganis, A. and M. Shanahan (2010). "Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity". In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, pp. 1–8.

Boyd, I. (1981). "The action of the three types of intrafusal fibre in isolated cat muscle spindles on the dynamic and length sensitivities of primary and secondary sensory endings". In: *Muscle receptors and movement*, pp. 17–32.

Brette, R. and W. Gerstner (2005). "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". In: *Journal of Neurophysiology* 94.5, pp. 3637–3642.

Briones, T. L., A. Y. Klintsova, and W. T. Greenough (2004). "Stability of synaptic plasticity in the adult rat visual cortex induced by complex environment exposure". In: *Brain research* 1018.1, pp. 130–135.

Broggi, A., A. Zelinsky, Ü. Özgüner, and C. Laugier (2016). "Intelligent vehicles". In: *Springer Handbook of Robotics*. Springer, pp. 1627–1656.

Brooks, R., L. Aryananda, A. Edsinger, P. Fitzpatrick, C. C. Kemp, U.-M. O'REILLY, E. Torres-Jara, P. Varshavskaya, and J. Weber (2004). "Sensing and manipulating built-for-human environments". In: *International Journal of Humanoid Robotics* 1.01, pp. 1–28.

Buchanan, T. S., D. G. Lloyd, K. Manal, and T. F. Besier (2004). "Neuromusculoskeletal modeling: estimation of muscle forces and joint moments and movements from measurements of neural command". In: *Journal of applied biomechanics* 20.4, pp. 367–395.

Burgess, N., J. G. Donnett, and J. O'Keefe (1998). "Using a mobile robot to test a model of the rat hippocampus". In: *Connection Science* 10.3-4, pp. 291–300.

Camacho, E. F. and C. B. Alba (2013). *Model predictive control*. Springer Science & Business Media.

Capurro, C., F. Panerai, and G. Sandini (1997). "Dynamic vergence using log-polar images". In: *International Journal of Computer Vision* 24.1, pp. 79–94.

Casellato, C., A. Antonietti, J. A. Garrido, R. R. Carrillo, N. R. Luque, E. Ros, A. Pedrocchi, and E. D'Angelo (2014). "Adaptive robotic control driven by a versatile spiking cerebellar network". In: *PLoS One* 9.11, e112265.

Cauli, N., E. Falotico, A. Bernardino, J. Santos-Victor, and C. Laschi (2016). "Correcting for changes: expected perception-based control for reaching a moving target". In: *IEEE Robotics & Automation Magazine* 23.1, pp. 63–70.

Chalmers, G. and P. Bawa (1997). "Synaptic connections from large afferents of wrist flexor and extensor muscles to synergistic motoneurones in man". In: *Experimental brain research* 116.2, pp. 351–358.

Chan, V., C. Jin, and A. van Schaik (2012). "Neuromorphic Audio-Visual Sensor Fusion on a Sound-Localising Robot". In: *Frontiers in Neuroscience* 6, p. 21.

Chen, W. and R. Poppele (1978). "Small-signal analysis of response of mammalian muscle spindles with fusimotor stimulation and a comparison with large-signal responses". In: *Journal of neurophysiology* 41.1, pp. 15–27.

Cisi, R. R. and A. F. Kohn (2008). "Simulation system of spinal cord motor nuclei and associated nerves and muscles, in a Web-based architecture". In: *Journal of computational neuroscience* 25.3, pp. 520–542.

Cofer, D., G. Cymbalyuk, W. J. Heitler, and D. H. Edwards (2010). "Control of tumbling during the locust jump". In: *Journal of Experimental Biology* 213.19, pp. 3378–3387.

Cofer, D., G. Cymbalyuk, J. Reid, Y. Zhu, W. J. Heitler, and D. H. Edwards (2010). "AnimatLab: a 3D graphics environment for neuromechanical simulations". In: *Journal of neuroscience methods* 187.2, pp. 280–288.

Collewijn, H., A. Martins, and R. Steinman (1981). "NATURAL RETINAL IMAGE MOTION: ORIGIN AND CHANGE". In: *Annals of the New York Academy of Sciences* 374.1, pp. 312–329.

Coumans, E. et al. (2013). "Bullet physics library". In: *Open source: bulletphysics.org*.

Cox, B. R. and J. L. Krichmar (2009). "Neuromodulation as a robot controller". In: *IEEE Robotics & Automation Magazine* 16.3.

Cullen, K. E. (2012). "The vestibular system: multimodal integration and encoding of self-motion for motor control". In: *Trends in neurosciences* 35.3, pp. 185–196.

Cullen, K. E. and J. E. Roy (2004). "Signal Processing in the Vestibular System During Active Versus Passive Head Movements". In: *Journal of Neurophysiology* 91.5, pp. 1919–1933.

Dacey, D. M. and O. S. Packer (2003). "Colour coding in the primate retina: diverse cell types and cone-specific circuitry". In: *Current opinion in neurobiology* 13.4, pp. 421–427.

Dallos, P. and R. Jones (1963). "Learning behavior of the eye fixation control system". In: *Automatic Control, IEEE Transactions on* 8.3, pp. 218–227.

Dario, P., M. C. Carrozza, E. Guglielmelli, C. Laschi, A. Menciassi, S. Micera, and F. Vecchi (2005). "Robotics as a future and emerging technology: biomimetics, cybernetics, and neuro-robotics in European projects". In: *Robotics & Automation Magazine, IEEE* 12.2, pp. 29–45.

Datteri, E., G. Teti, C. Laschi, G. Tamburrini, G. Dario, and E. Guglielmelli (2003). "Expected perception: an anticipation-based perception-action scheme in robots". In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 1. IEEE, pp. 934–939.

Davison, A. ., D. Brüderle, J. M. Eppler, J. Kremkow, E. Muller, D. A. Pecevski, L. Perrinet, and P. Yger (2008). "PyNN: a common interface for neuronal network simulators". In: *Front. Neuroinform.*

Delp, S. L., F. C. Anderson, A. S. Arnold, P. Loan, A. Habib, T. John, E. Guendelman, and D. G. Thelen (2007). "OpenSim: Open-source Software to Create and Analyze Dynamic Simulations of Movement". In: *IEEE transactions on biomedical engineering* 54.11, pp. 1940–1950.

Denoyelle, N., F. Pouget, T. Viéville, and F. Alexandre (2014). "VirtualEnaction: A Platform for Systemic Neuroscience Simulation." In: *International Congress on Neurotechnology, Electronics and Informatics*.

Diedam, H., D. Dimitrov, P.-B. Wieber, K. Mombaur, and M. Diehl (2008). "Online walking gait generation with adaptive foot positioning through linear model predictive control". In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, pp. 1121–1126.

Djurfeldt, M., J. Hjorth, J. M. Eppler, N. Dudani, M. Helias, T. C. Potjans, U. S. Bhalla, M. Diesmann, J. H. Kotaleski, and Ö. Ekeberg (2010). "Run-time interoperability between neuronal network simulators based on the MUSIC framework". In: *Neuroinformatics* 8.1, pp. 43–60.

Drumwright, E. and al. (2010). "Extending Open Dynamics Engine for robotics simulation". In: *Simulation, Modeling, and Programming for Autonomous Robots*. Vol. 6472. Lecture Notes in Computer Science. Springer, pp. 38–50.

Elias, L. A., R. N. Watanabe, and A. F. Kohn (2014). "Spinal mechanisms may provide a combination of intermittent and continuous control of human posture: predictions from a biologically based neuromusculoskeletal model". In: *PLoS Comput Biol* 10.11, e1003944.

Ellaway, P. H., A. Taylor, and R. Durbaba (2015). "Muscle spindle and fusimotor activity in locomotion". In: *Journal of anatomy* 227.2, pp. 157–166.

Fallon, M., S. Kuindersma, S. Karumanchi, M. Antone, T. Schneider, H. Dai, C. P. D'Arpino, R. Deits, M. DiCicco, D. Fourie, et al. (2015). "An Architecture for Online Affordance-based Perception and Whole-body Planning". In: *Journal of Field Robotics* 32.2, pp. 229–254.

Falotico, E., N. Cauli, K. Hashimoto, P. Kryczka, A. Takanishi, P. Dario, A. Berthoz, and C. Laschi (Sept. 2012). "Head stabilization based on a feedback error learning in a humanoid robot". In: *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication, 2012 IEEE International Conference on*, pp. 449–454.

Falotico, E., N. Cauli, P. Kryczka, K. Hashimoto, A. Berthoz, A. Takanishi, P. Dario, and C. Laschi (2017). "Head stabilization in a humanoid robot: models and implementations". In: *Autonomous Robots* 41.2, pp. 349–365.

Falotico, E., C. Laschi, P. Dario, D. Bernardin, and A. Berthoz (2011). "Using trunk compensation to model head stabilization during locomotion". In: *IEEE-RAS International Conference on Humanoid Robots*, pp. 440–445.

Falotico, E., M. Taiana, D. Zambrano, A. Bernardino, J. Santos-Victor, P. Dario, and C. Laschi (2009). "Predictive tracking across occlusions in the iCub robot". In: *Proceedings of the 9th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2009)*, pp. 486–491.

Falotico, E., D. Zambrano, G. G. Muscolo, L. Marazzato, P. Dario, and C. Laschi (2010). "Implementation of a bio-inspired visual tracking model on the iCub robot". In: *Proc. 19th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN'10)*. IEEE, pp. 564–569.

Fitzpatrick, P., K. Harada, C. C. Kemp, Y. Matsumoto, K. Yokoi, and E. Yoshida (2016). "Humanoids". In: *Springer Handbook of Robotics*. Springer, pp. 1789–1818.

Folgheraiter, M. and G. Gini (2004). "Human-like reflex control for an artificial hand". In: *BioSystems* 76.1, pp. 65–74.

Forbes, P. A., E. de Bruijn, A. C. Schouten, F. C. van der Helm, and R. Happee (2013). "Dependency of human neck reflex responses on the bandwidth of pseudorandom anterior-posterior torso perturbations". In: *Experimental brain research* 226.1, pp. 1–14.

Forbes, P. A., G. P. Siegmund, A. C. Schouten, and J.-S. Blouin (2014). "Task, muscle and frequency dependent vestibular control of posture". In: *Frontiers in integrative neuroscience* 8.

Fox, C. W., M. D. Humphries, B. Mitchinson, T. Kiss, Z. Somogyva, and T. J. Prescott (2009). "Technical integration of hippocampus, basal ganglia and physical models for spatial navigation". In: *Frontiers in neuroinformatics* 3, p. 6.

Franchi, E., E. Falotico, D. Zambrano, G. Muscolo, L. Marazzato, P. Dario, and C. Laschi (2010). "A comparison between two bio-inspired adaptive models of Vestibulo-Ocular Reflex (VOR) implemented on the iCub robot". In: *Proceedings of the 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2010)*, pp. 251–256.

Fritzke (1995). "A growing neural gas network learns topologies". In: *Advances in neural information processing systems* 7, pp. 625–632.

Fuglevand, A. J., D. A. Winter, and A. E. Patla (1993). "Models of recruitment and rate coding organization in motor-unit pools". In: *Journal of neurophysiology* 70.6, pp. 2470–2488.

Fukuda, T., P. Dario, and G.-Z. Yang (2017). "Humanoid robotics—History, current state of the art, and challenges". In: *Science Robotics* 2.13.

Fukushima, J., N. Morita, K. Fukushima, T. Chiba, S. Tanaka, and I. Yamashita (1990). "Voluntary control of saccadic eye movements in patients with schizophrenic and affective disorders". In: *Journal of Psychiatric Research* 24.1, pp. 9–24.

Galiana, H. L. and J. S. Outerbridge (1984). "A bilateral model for central neural pathways in vestibuloocular reflex". In: *Journal of neurophysiology* 51.2, pp. 210–241.

Gamez, D., A. K. Fidjeland, and E. Lazdins (2012). "iSpike: a spiking neural interface for the iCub robot". In: *Bioinspiration Biomimetics* 7.2, p. 025008.

Gamez, D., R. Newcombe, O. Holland, and R. Knight (2006). "Two simulation tools for biologically inspired virtual robotics". In: *Proceedings of the IEEE 5th Chapter Conference on Advances in Cybernetic Systems*, pp. 85–90.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Gay, S., J. Santos-Victor, and A. Ijspeert (Nov. 2013). "Learning robot gait stability using neural networks as sensory feedback function for Central Pattern Generators". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 194–201.

Gewaltig, M.-O. and M. Diesmann (2007). "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4, p. 1430.

Goldberg, J. M. and K. E. Cullen (2011). "Vestibular control of the head: possible functions of the vestibulocollic reflex". In: *Experimental brain research* 210.3-4, pp. 331–345.

Gomez-Rodriguez, F., A. Linares-Barranco, L. Miro, S.-C. Liu, A. van Schaik, R. Etienne-Cummings, and M. A. Lewis (2007). "AER auditory filtering and CPG for robot control". In: *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*. IEEE, pp. 1201–1204.

Gomi, H. and M. Kawato (1992). "Adaptive feedback control models of the vestibulocerebellum and spinocerebellum". In: *Biological cybernetics* 68.2, pp. 105–114.

Goodman, D. and R. Brette (2008). "Brian: a simulator for spiking neural networks in Python". In:

Green, A. M. and D. E. Angelaki (2004). "An integrative neural network for detecting inertial motion and head orientation". In: *Journal of neurophysiology* 92.2, pp. 905–925.

Gross, H.-M., A. Heinze, T. Seiler, and V. Stephan (1999). "Generative character of perception: A neural architecture for sensorimotor anticipation". In: *Neural Networks* 12.7-8, pp. 1101–1129.

Grossberg, S. (1976). "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors". In: *Biological cybernetics* 23.3, pp. 121–134.

Guizzo, E. and E. Ackerman (2015). "The hard lessons of darpa's robotics challenge [news]". In: *IEEE Spectrum* 52.8, pp. 11–13.

Habra, T. and R. Ronsse (2016). "Gaze stabilization of a humanoid robot based on virtual linkage". In: *Biomedical Robotics and Biomechatronics (BioRob), 2016 6th IEEE International Conference on*. IEEE, pp. 163–169.

Hasan, Z. (1983). "A model of spindle afferent response to muscle stretch". In: *Journal of neurophysiology* 49.4, pp. 989–1006.

Hashimoto, K., H.-J. Kang, M. Nakamura, E. Falotico, H.-O. Lim, A. Takanishi, C. Laschi, P. Dario, and A. Berthoz (Oct. 2012). "Realization of biped walking on soft ground with stabilization control based on gait analysis". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 2064–2069.

Heeger, D. (2000). "Poisson model of spike generation". In: *Handout, University of Standford* 5, pp. 1–13.

Henze, B., C. Ott, and M. A. Roa (2014). "Posture and balance control for humanoid robots in multi-contact scenarios based on model predictive control". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, pp. 3253–3258.

Hinkel, G., H. Groenda, L. Vannucci, O. Denninger, N. Cauli, and S. Ulbrich (2015). "A domain-specific language (DSL) for integrating neuronal networks in robot control". In: *ACM International Conference Proceeding Series*. Vol. 21-July-2015, pp. 9–15.

Hinkel, G., H. Groenda, S. Krach, L. Vannucci, O. Denninger, N. Cauli, S. Ulbrich, A. Roennau, E. Falotico, M.-O. Gewaltig, et al. (2016). "A Framework for Coupled Simulations of Robots and Spiking Neuronal Networks". In: *Journal of Intelligent & Robotic Systems*, in press.

Hoffmann, H. (2007). "Perception through visuomotor anticipation in a mobile robot". In: *Neural Networks* 20.1, pp. 22–33.

Houk, J. C., W. Z. Rymer, and P. E. Crago (1981). "Dependence of dynamic response of spindle receptors on muscle length and velocity". In: *Journal of Neurophysiology* 46.1, pp. 143–166.

IEEE (Oct. 1998). "IEEE Recommended Practice for Software Requirements Specifications". In: *IEEE Std 830-1998*, pp. 1–40.

Iida, F. and A. J. Ijspeert (2016). "Biologically inspired robotics". In: *Springer Handbook of Robotics*. Springer, pp. 2015–2034.

Issa, F. A., J. Drummond, D. Cattaert, and D. H. Edwards (2012). "Neural circuit reconfiguration by social status". In: *The Journal of Neuroscience* 32.16, pp. 5638–5645.

Ito, M. (1984). *The cerebellum and neural control*. Raven Pr.

— (2000). "Mechanisms of motor learning in the cerebellum". In: *Brain Research* 886.1–2, pp. 237–245.

— (2012). *The cerebellum: brain for an implicit self*. FT press.

Ito, M., N. Nisimaru, and M. Yamamoto (1977). "Specific patterns of neuronal connexions involved in the control of the rabbit's vestibulo-ocular reflexes by the cerebellar flocculus." In: *The Journal of Physiology* 265.3, pp. 833–854.

Janabi-Sharifi, F., V. Hayward, and C.-S. Chen (2000). "Discrete-time adaptive windowing for velocity estimation". In: *IEEE Transactions on Control Systems Technology* 8.6, pp. 1003–1009.

Johansson, R. S. (1998). "Sensory input and control of grip". In: *Sensory guidance of movement. Novartis foundation symposium*. Vol. 218, pp. 45–59.

Jones, J. L. (2006). "Robots at the tipping point: the road to iRobot Roomba". In: *IEEE Robotics & Automation Magazine* 13.1, pp. 76–78.

Kaji, R., J. C. Rothwell, M. Katayama, T. Ikeda, T. Kubori, N. Kohara, T. Mezaki, H. Shibasaki, and J. Kimura (1995). "Tonic vibration reflex and muscle afferent block in writer's cramp". In: *Annals of neurology* 38.2, pp. 155–162.

Kaminaga, H., T. Amari, Y. Katayama, J. Ono, Y. Shimoyama, and Y. Nakamura (2010). "Backdrivability analysis of electro-hydrostatic actuator and series dissipative actuation model". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, pp. 4204–4211.

Kandel, E. R., J. H. Schwartz, T. M. Jessell, S. A. Siegelbaum, A. J. Hudspeth, et al. (2000). *Principles of neural science*. Vol. 4. McGraw-hill New York.

Kang, H.-J., K. Hashimoto, K. Nishikawa, E. Falotico, H.-O. Lim, A. Takanishi, C. Laschi, P. Dario, and A. Berthoz (June 2012). "Biped walking stabilization on

soft ground based on gait analysis". In: *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS EMBS International Conference on*, pp. 669–674.

Kavanagh, J., R. Barrett, and S. Morrison (2006). "The role of the neck and trunk in facilitating head stability during walking". In: *Experimental brain research* 172.4, pp. 454–463.

Khan, M. M., D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber (2008). "SpiNNaker: mapping neural networks onto a massively-parallel chip multiprocessor". In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, pp. 2849–2856.

Kim, S. and A. Billard (2012). "Estimating the non-linear dynamics of free-flying objects". In: *Robotics and Autonomous Systems* 60.9, pp. 1108–1122.

Knoll, A. and M.-O. Gewaltig (2016). "Neurorobotics: A strategic pillar of the Human Brain Project". In: *Brain-inspired intelligent robotics: The intersection of robotics and neuroscience*, pp. 25–34.

Kober, J., M. Glisson, and M. Mistry (2012). "Playing catch and juggling with a humanoid robot". In: *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, pp. 875–881.

Koenemann, J., A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard (2015). "Whole-body model-predictive control applied to the HRP-2 humanoid". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, pp. 3346–3351.

Koenig, N. and A. Howard (2004). "Design and use paradigms for gazebo, an open-source multi-robot simulator". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE, pp. 2149–2154.

Kohonen, T. (2001). *Self-organizing maps*. Vol. 30. Springer.

Kokotovie, P. V. (1992). "The joy of feedback: nonlinear and adaptive". In: *IEEE Control Systems Magazine* 12.3, pp. 7–17.

Krauzlis, R. J. and S. G. Lisberger (1994). "A model of visually-guided smooth pursuit eye movements based on behavioral observations". In: *Journal of computational neuroscience* 1.4, pp. 265–283.

Krotkov, E., D. Hackett, L. Jackel, M. Perschbacher, J. Pippine, J. Strauss, G. Pratt, and C. Orlowski (2017). "The DARPA robotics challenge finals: results and perspectives". In: *Journal of Field Robotics* 34.2, pp. 229–240.

Kryczka, P., E. Falotico, K. Hashimoto, H.-O. Lim, A. Takanishi, C. Laschi, P. Dario, and A. Berthoz (2012). "A robotic implementation of a bio-inspired head motion stabilization model on a humanoid platform". In: *IEEE International Conference on Intelligent Robots and Systems*, pp. 2076–2081.

Kryczka, P., E. Falotico, K. Hashimoto, H. Lim, A. Takanishi, C. Laschi, P. Dario, and A. Berthoz (2012). "Implementation of a human model for head stabilization on a humanoid platform". In: *Biomedical Robotics and Biomechatronics (BioRob), 2012 4th IEEE RAS & EMBS International Conference on*. IEEE, pp. 675–680.

Kunkel, S., M. Schmidt, J. M. Eppler, H. E. Plesser, G. Masumoto, J. Igarashi, S. Ishii, T. Fukai, A. Morrison, M. Diesmann, and M. Helias (2014). "Spiking network simulation code for petascale computers". In: *Frontiers in Neuroinformatics* 8.78.

Laschi, C., G. Asuni, E. Guglielmelli, G. Teti, R. Johansson, H. Konosu, Z. Wasik, M. C. Carrozza, and P. Dario (2008). "A bio-inspired predictive sensory-motor coordination scheme for robot reaching and preshaping". In: *Autonomous Robots* 25.1-2, pp. 85–101.

Laschi, C., G. Asuni, G. Teti, M. C. Carrozza, P. Dario, E. Guglielmelli, and R. Johansson (2006). "A bio-inspired neural sensory-motor coordination scheme for

robot reaching and preshaping". In: *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*. IEEE, pp. 531–536.

Lau, C. G., V. Honrubia, H. Jenkins, R. Baloh, and R. Yee (1978). "Linear model for visual-vestibular interaction." In: *Aviation, space, and environmental medicine*.

Lenz, A., T. Balakrishnan, A. G. Pipe, and C. Melhuish (2008). "An adaptive gaze stabilization controller inspired by the vestibulo-ocular reflex". In: *Bioinspiration & Biomimetics* 3.3, p. 035001.

Lim, H.-O., Y. Ogura, and A. Takanishi (Oct. 2006). "Dynamic Locomotion and Mechanism of Biped Walking Robot". In: *2006 SICE-ICASE International Joint Conference*, pp. 3484–3489.

Lim, H.-O., Y. Kaneshima, and A. Takanishi (2002). "Online walking pattern generation for biped humanoid robot with trunk". In: *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 3. IEEE, pp. 3111–3116.

Lim, H.-O., Y. Yamamoto, and A. Takanishi (2002). "Stabilization control for biped follow walking". In: *Advanced Robotics* 16.4, pp. 361–380.

Lin, C.-C. K. and P. E. Crago (2002). "Structural model of the muscle spindle". In: *Annals of biomedical engineering* 30.1, pp. 68–83.

Lucia, S., T. Finkler, and S. Engell (2013). "Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty". In: *Journal of Process Control* 23.9, pp. 1306–1319.

Lund, H. H., B. Webb, and J. Hallam (1998). "Physical and temporal scaling considerations in a robot model of cricket calling song preference". In: *Artificial life* 4.1, pp. 95–107.

Luque, N. R., J. A. Garrido, R. R. Carrillo, S. Tolu, and E. Ros (2011). "Adaptive cerebellar spiking model embedded in the control loop: context switching and robustness against noise". In: *International Journal of Neural Systems* 21.05, pp. 385–401.

Maltenfort, M. G. and R. Burke (2003). "Spindle model responsive to mixed fusimotor inputs and testable predictions of β feedback effects". In: *Journal of neurophysiology* 89.5, pp. 2797–2809.

Manzoni, D., P. Andre, P. d'Ascanio, and O. Pompeiano (Oct. 1994). "Depression of the vestibulospinal reflex adaptation by intravermal microinjection of GABA-A and GABA-B agonists in the cat". In: *Archives italiennes de biologie* 132.4, pp. 243–269.

Marcinkiewicz, M., R. Kaushik, I. Labutov, S. Parsons, and T. Raphan (2009). "Learning to stabilize the head of a quadrupedal robot with an artificial vestibular system". In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, pp. 2512–2517.

Marr, D. and W. T. Thach (1991). "From the Retina to the Neocortex: Selected Papers of David Marr". In: ed. by L. Vaina. Boston, MA: Birkhäuser Boston. Chap. A Theory of Cerebellar Cortex, pp. 11–50.

Martinetz, T., K. Schulten, et al. (1991). *A" neural-gas" network learns topologies*. University of Illinois at Urbana-Champaign.

Martínez-Cañada, P., C. Morillas, J. L. Nieves, B. Pino, and F. Pelayo (2015). "First Stage of a Human Visual System Simulator: The Retina". In: *Computational Color Imaging*. Springer, pp. 118–127.

Martínez-Cañada, P., C. Morillas, B. Pino, E. Ros, and F. Pelayo (2016). "A COMPUTATIONAL FRAMEWORK FOR REALISTIC RETINA MODELING". In: *International Journal of Neural Systems* 26.07, p. 1650030.

Matthews, P. and R. Stein (1969). "The sensitivity of muscle spindle afferents to small sinusoidal changes of length". In: *The Journal of Physiology* 200.3, p. 723.

Metta, G., P. Fitzpatrick, and L. Natale (2006). "YARP: Yet another robot platform". In: *International Journal of Advanced Robotic Systems* 3.1. cited By 185, pp. 043–048.

Metta, G., G. Sandini, D. Vernon, L. Natale, and F. Nori (2008). "The iCub humanoid robot: an open platform for research in embodied cognition". In: *Proceedings of the 8th workshop on performance metrics for intelligent systems*. ACM, pp. 50–56.

Miall, R. C. and D. M. Wolpert (1996). "Forward models for physiological motor control". In: *Neural networks* 9.8, pp. 1265–1279.

Miles, F. A. and S. G. Lisberger (1981). "Plasticity in the vestibulo-ocular reflex: a new hypothesis". In: *Annual review of neuroscience* 4.1, pp. 273–299.

Mileusnic, M. P., I. E. Brown, N. Lan, and G. E. Loeb (2006). "Mathematical models of proprioceptors. I. Control and transduction in the muscle spindle". In: *Journal of neurophysiology* 96.4, pp. 1772–1788.

Miyamoto, H., M. Kawato, T. Setoyama, and R. Suzuki (1988). "Feedback-error-learning neural network for trajectory control of a robotic manipulator". In: *Neural networks* 1.3, pp. 251–265.

Moraud, E. M., M. Capogrosso, E. Formento, N. Wenger, J. DiGiovanna, G. Courtine, and S. Micera (2016). "Mechanisms underlying the neuromodulation of spinal circuits for correcting gait and balance deficits after spinal cord injury". In: *Neuron* 89.4, pp. 814–828.

Moutinho, N., N. Cauli, E. Falotico, R. Ferreira, J. Gaspar, A. Bernardino, J. Santos-Victor, P. Dario, and C. Laschi (2011). "An expected perception architecture using visual 3d reconstruction for a humanoid robot". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, pp. 4826–4831.

Mulas, M., M. Zhan, and J. Conradt (2015). "Integration of biological neural models for the control of eye movements in a robotic head". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9222, pp. 231–242.

Mulavara, A. P., M. C. Verstraete, and J. J. Bloomberg (2002). "Modulation of head movement control in humans during treadmill walking". In: *Gait & posture* 16.3, pp. 271–282.

Muscolo, G., C. Recchiuto, K. Hashimoto, P. Dario, and A. Takanishi (2012). "Towards an improvement of the SABIAN humanoid robot: From design to optimization". In: *Journal of Mechanical Engineering and Automation* 2.4, pp. 80–84.

Nakamura, Y., K. Yamane, Y. Fujita, and I. Suzuki (2005). "Somatosensory computation for man-machine interface from motion-capture data and musculoskeletal human model". In: *IEEE Transactions on Robotics* 21.1, pp. 58–66.

Nakanishi, Y., Y. Asano, T. Kozuki, H. Mizoguchi, Y. Motegi, M. Osada, T. Shirai, J. Urata, K. Okada, and M. Inaba (2012). "Design concept of detail musculoskeletal humanoid "Kenshiro" - Toward a real human body musculoskeletal simulator". In: *Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on*. IEEE, pp. 1–6.

Nguyen-Tuong, D. and J. Peters (2011). "Model learning for robot control: a survey". In: *Cognitive processing* 12.4, pp. 319–340.

Niu, C. M., K. Jalaleddini, W. J. Sohn, J. Rocamora, T. D. Sanger, and F. J. Valero-Cuevas (2017). "Neuromorphic meets neuromechanics, part I: the methodology and implementation". In: *Journal of Neural Engineering* 14.2, p. 025001.

Ogura, Y., H. Aikawa, K. Shimomura, H. Kondo, A. Morishima, H.-o. Lim, and A. Takanishi (2006). "Development of a new humanoid robot WABIAN-2". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, pp. 76–81.

Otten, E., K. Scheepstra, and M. Hulliger (1995). "An integrated model of the mammalian muscle spindle". In: *Alpha and Gamma Motor Systems*. Springer, pp. 294–301.

Panerai, F., G. Metta, and G. Sandini (2000). "Visuo-inertial stabilization in space-variant binocular systems". In: *Robotics and Autonomous Systems* 30.1, pp. 195–214.

Panerai, F. and G. Sandini (1998). "Oculo-motor stabilization reflexes: integration of inertial and visual information". In: *Neural Networks* 11.7, pp. 1191–1204.

Peng, G., T. Hain, and B. Peterson (1996). "A dynamical model for reflex activated head movements in the horizontal plane". In: *Biological cybernetics* 75.4, pp. 309–319.

Peterson, B. W., H. Choi, T. Hain, E. Keshner, and G. C. Peng (2001). "Dynamic and kinematic strategies for head movement control". In: *Annals of the New York Academy of Sciences* 942.1, pp. 381–393.

Pierrot-Deseilligny, E. and D. Burke (2005). *The circuitry of the human spinal cord: its role in motor control and movement disorders*. Cambridge University Press.

Porrill, J. and P. Dean (2007). "Recurrent Cerebellar Loops Simplify Adaptive Control of Redundant and Nonlinear Motor Systems". In: *Neural Computation* 19.1, pp. 170–193.

Porrill, J., P. Dean, and J. V. Stone (2004). "Recurrent cerebellar architecture solves the motor-error problem". In: *Proceedings of the Royal Society of London-B* 271.1541, pp. 789–796.

Potjans, T. C. and M. Diesmann (2014). "The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking network model". In: *Cerebral Cortex* 24.3, pp. 785–806.

Pozzo, T., A. Berthoz, and L. Lefort (1990). "Head stabilization during various locomotor tasks in humans". In: *Experimental Brain Research* 82.1, pp. 97–106.

Prassler, E., M. E. Munich, P. Pirjanian, and K. Kosuge (2016). "Domestic robotics". In: *Springer handbook of robotics*. Springer, pp. 1729–1758.

Prochazka, A. and M. Gorassini (1998). "Models of ensemble firing of muscle spindle afferents recorded during normal locomotion in cats". In: *The Journal of physiology* 507.1, pp. 277–291.

Proske, U. (1997). "The mammalian muscle spindle". In: *Physiology* 12.1, pp. 37–42.

Quigley, M., K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng (2009). "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*, p. 5.

Reeke, G. N., O. Sporns, and G. M. Edelman (1990). "Synthetic neural modeling: the 'Darwin' series of recognition automata". In: *Proceedings of the IEEE* 78.9, pp. 1498–1530.

Richter, C., S. Jentzsch, R. Hostettler, J. A. Garrido, E. Ros, A. Knoll, F. Rohrbein, P. van der Smagt, and J. Conradt (2016). "Musculoskeletal Robots: Scalability in Neural Control". In: *IEEE Robotics & Automation Magazine* 23.4, pp. 128–137.

Robinson, D. A., J. Gordon, and S. Gordon (1986). "A model of the smooth pursuit eye movement system". In: *Biological cybernetics* 55.1, pp. 43–57.

Robinson, D. (1976). "Adaptive gain control of vestibuloocular reflex by the cerebellum". In: *Journal of Neurophysiology* 39.5, pp. 954–969.

Roncone, A., U. Pattacini, G. Metta, and L. Natale (2014). "Gaze stabilization for humanoid robots: A comprehensive framework". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, pp. 259–264.

Ros, E., E. M. Ortigosa, R. Carrillo, and M. Arnold (2006). "Real-time computing platform for spiking neurons (RT-spike)". In: *Neural Networks, IEEE Transactions on* 17.4, pp. 1050–1063.

Rosenblatt, F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Sandini, G., F. Panerai, and F. A. Miles (2001). "The role of inertial and visual mechanisms in the stabilization of gaze in natural and artificial systems". In: *Motion Vision*. Springer, pp. 189–218.

Schmid, R., A. Buizza, and D. Zambarbieri (1980). "A non-linear model for visual-vestibular interaction during body rotation in man". In: *Biological cybernetics* 36.3, pp. 143–151.

Schmid, R., M. Stefanelli, and E. Mira (1971). "Mathematical modelling: a contribution to clinical vestibular analysis". In: *Acta oto-laryngologica* 72.1-6, pp. 292–302.

Schwaber, K. and M. Beedle (2002). "Agile Software Development with Scrum". In:

Schweigart, G., T. Mergner, I. Evdokimidis, S. Morand, and W. Becker (1997). "Gaze stabilization by optokinetic reflex (OKR) and vestibulo-ocular reflex (VOR) during active head rotation in man". In: *Vision Research* 37.12, pp. 1643–1652.

Scokaert, P. O. and D. Mayne (1998). "Min-max feedback model predictive control for constrained linear systems". In: *IEEE Transactions on Automatic control* 43.8, pp. 1136–1142.

Shibata, T., H. Tabata, S. Schaal, and M. Kawato (2005). "A model of smooth pursuit in primates based on learning the target dynamics". In: *Neural Networks* 18.3, pp. 213–224.

Shibata, T., S. Vijayakumar, J. Conradt, and S. Schaal (2001). "Biomimetic oculomotor control". In: *Adaptive Behavior* 9.3-4, pp. 189–207.

Sreenivasa, M., K. Ayusawa, and Y. Nakamura (2016). "Modeling and identification of a realistic spiking neural network and musculoskeletal model of the human arm, and an application to the stretch reflex". In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 24.5, pp. 591–602.

Stewart, T. C., A. Kleinhans, A. Mundy, and J. Conradt (2016). "Serendipitous offline learning in a neuromorphic robot". In: *Frontiers in neurorobotics* 10.

Stienen, A. H., A. C. Schouten, J. Schuurmans, and F. C. Van Der Helm (2007). "Analysis of reflex modulation with a biologically realistic neural network". In: *Journal of computational neuroscience* 23.3, p. 333.

Stroeve, S. (1999). "Impedance characteristics of a neuromusculoskeletal model of the human arm I. Posture control". In: *Biological cybernetics* 81.5, pp. 475–494.

Takeichi, K. (2015). "Neuro-musculoskeletal Simulator with anatomically and Physiologically based Spiking Neuron Spine Model". MA thesis. University of Tokyo.

Tikhanoff, V., A. Cangelosi, P. Fitzpatrick, G. Metta, L. Natale, and F. Nori (2008). "An Open-source Simulator for Cognitive Robotics Research: The Prototype of the iCub Humanoid Robot Simulator". In: *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems*. PerMIS '08. Gaithersburg, Maryland: ACM, pp. 57–61.

Tolu, S., M. Vanegas, J. A. Garrido, N. R. Luque, and E. Ros (2013). "Adaptive and predictive control of a simulated robot arm". In: *Int. J. Neural Syst.* 23.3.

Tolu, S., M. Vanegas, N. R. Luque, J. A. Garrido, and E. Ros (2012). "Bio-inspired adaptive feedback error learning architecture for motor control". In: *Biological Cybernetics* 106.8-9, pp. 507–522.

Traver, V. J. and A. Bernardino (2010). "A review of log-polar imaging for visual perception in robotics". In: *Robotics and Autonomous Systems* 58.4, pp. 378–398.

Tsagarakis, N. G., D. G. Caldwell, A. Bicchi, F. Negrello, M. Garabini, W. Choi, L. Baccelliere, V. Loc, J. Noorden, M. Catalano, et al. (2016). "WALK-MAN: A high performance humanoid platform for realistic environments". In: *Journal of Field Robotics (JFR)*.

Vannucci, L., N. Cauli, E. Falotico, A. Bernardino, and C. Laschi (2014). "Adaptive visual pursuit involving eye-head coordination and prediction of the target motion". In: *Proceedings of the 14th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2014)*. IEEE, pp. 541–546.

Vasco, V., A. Glover, Y. Tirupachuri, F. Solari, M. Chessa, and C. Bartolozzi (2016). "Vergence control with a neuromorphic iCub". In: *IEEE-RAS International Conference on Humanoid Robots*. IEEE, pp. 732–738.

Vijayakumar, S. and S. Schaal (2000). "Locally Weighted Projection Regression: Incremental Real Time Learning in High Dimensional Space". In: *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 1079–1086.

Viollet, S. and N. Franceschini (2005). "A high speed gaze control system based on the vestibulo-ocular reflex". In: *Robotics and Autonomous systems* 50.4, pp. 147–161.

Voegtlin, T. (2011). "CLONES: a closed-loop simulation framework for body, muscles and neurons". In: *BMC Neuroscience* 12, pp. 1–1.

Weidel, P., M. Djurfeldt, R. C. Duarte, and A. Morrison (2016). "Closed Loop Interactions between Spiking Neural Network and Robotic Simulators Based on MUSIC and ROS". In: *Frontiers in Neuroinformatics* 10, p. 31.

Weidel, P., R. Duarte, K. Korvasová, J. Jitsev, and A. Morrison (2015). "ROS-MUSIC toolchain for spiking neural network simulations in a robotic environment". In: *BMC Neuroscience* 16.1, p. 1.

Weigend, A. S., B. A. Huberman, and D. E. Rumelhart (1990). "Predicting the future: A connectionist approach". In: *International journal of neural systems* 1.03, pp. 193–209.

Wells, S. and G. R. Barnes (1998). "Fast, anticipatory smooth-pursuit eye movements appear to depend on a short-term store". In: *Experimental brain research* 120.1, pp. 129–133.

Whittaker, S. G. and G. Eaholtz (1982). "Learning patterns of eye motion for foveal pursuit." In: *Investigative ophthalmology & visual science* 23.3, pp. 393–397.

Widrow, B. and M. E. Hoff (1960). "Adaptive switching circuits." In:

Wieber, P.-B. (2006). "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations". In: *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*. IEEE, pp. 137–142.

Wolpert, D. M., R. C. Miall, and M. Kawato (1998). "Internal models in the cerebellum". In: *Trends in cognitive sciences* 2.9, pp. 338–347.

Xivry, J.-J. O. de, S. Coppe, G. Blohm, and P. Lefevre (2013). "Kalman filtering naturally accounts for visually guided and predictive smooth pursuit dynamics". In: *The Journal of Neuroscience* 33.44, pp. 17301–17313.

Yamada, H., M. Mori, and S. Hirose (2007). "Stabilization of the head of an undulating snake-like robot". In: *Intelligent Robots and Systems (IROS), 2007 IEEE/RSJ International Conference on*, pp. 3566–3571.

Yamaguchi, J., E. Soga, S. Inoue, and A. Takanishi (1999). "Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 1. IEEE, pp. 368–374.

Zambrano, D., E. Falotico, L. Manfredi, and C. Laschi (2010). "A model of the smooth pursuit eye movement with prediction and learning". In: *Applied Bionics and Biomechanics* 7.2, pp. 109–118.